AN OPERATING SYSTEM MODEL FOR REAL-TIME APPLICATIONS

William Charles Regmund

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

An Operating System Model for Real-Time Applications

by

William Charles Regmund, Jr.

Thesis Advisor: G.L. Barksdale

December 1972

An Operating System Model for Real-Time Applications

by

William Charles Regmund, Jr.
Lieutenant, United States Navy
B.A., Rice University, 1965
B.S., Rice University, 1966

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1972

ABSTRACT

The basic organization of an operating system is often obscured by the myriad details of a particular implementation. In any context where the conceptual behavior of such systems is of interest, it is therefore desirable to have at hand some device which is transparent to fundamental concepts but opaque to the details. The purpose of this work is to develop such a device in the form of a logical model of certain operating system functions. In order to test the utility of this model, it is translated into a computer simulation program. Since operating systems for real-time applications are of particular interest here, some features of real-time systems that are built into the computer model are discussed. Finally, some applications of the program, and thus of the logical model, are suggested.

TABLE OF CONTENTS

I. INTRODUCTION

The principal concern of this research is the modeling of operating systems for real-time computing environments. The particular model developed herein is designed to study those sections of an operating system devoted to resource scheduling; these sections will be referred to as schedulers. Attention is given at various point throughout the paper to the relation of multiprocessing concepts to the basic model; the goal is to develop the model in such a way that its usefulness is not restricted to systems with a single processor. The end result of this work is a computer program which implements the elements of the logical model and which can be used to study a certain class of real-time applications.

The work divides naturally into three major areas. The first is concerned with the development of a logically streamlined viewpoint of schedulers. The problem of describing a program as complex as an operating system is introduced and some desirable characteristics for a model are discussed. The next three sections of the paper define the main entities of a particular model which has rather general application to the study of multiprocessing schedulers. Extensions to the basic model are taken up next and a particular example is discussed.

The second area for consideration is the real-time domain. Some properties of real-time systems are discussed in order to focus attention on a particular class of such systems. The class of interest is characterized by multiple, complex timing constraints found typically in process control, data acquisition and command and control

applications. Details of a specific Naval system of this class, the MK 86 MOD 2 Gun Fire Control System, are discussed in Chapter III, Section B. The purpose in this discussion is to pave the way for the definition of a hypothetical real-time system (similar to the MK 86 MOD 2) which is actually modeled by the computer program. Various assumptions and characteristics of the hypothetical system are reviewed.

Finally, the first two areas are brought together and mapped onto an implementation in the form of a program. The general make-up of the program is discussed. Specific information on the implementation of logical model entities is given. The several Appendices describe details of the program design and utilization.

II. THE LOGICAL MODEL

A. SIMPLIFYING VIEWPOINTS

The description of a modern operating system is a difficult job because such a system has a large number of tasks to perform and most of them seem to be quite intricate and delicately balanced. Even a major subsystem, e.g., resource scheduling, generally follows this same pattern. Typically, for each minute sub-task that is considered, disproportionate numbers of implementation questions arise to retard the progress of the design. In this section we will attempt to reduce operating systems descriptions to a set of more manageable essentials. The goal is a set of distinct logical blocks into which these essential elements fit so that every function of interest can be simply and uniquely located in some block. It is this set of blocks that form the "logical model". Since one of the overall goals of this project involves integration of multiprocessing concepts, some clarification of the term "multiprocessing" is in order. Parallel processing is discussed in much of the current literature [6]; multiprocessing concepts are often found embedded in this context. The multiprocessing systems referred to in this report do not attempt to subdivide a given job into segments that may be run in parallel. They have only a small number of processors (typically 2 or 3) and are thus not of the same class as the highly parallel ILLIAC IV or PEPE systems [6].

A simplified way of thinking about multiprocessing systems is desirable in view of their apparent complexity. By design such a schema is likely to be slanted toward a particular aspect of the system. Although a particular logical way of looking at real-time operating systems is thus nominally designed for a particular purpose, it should

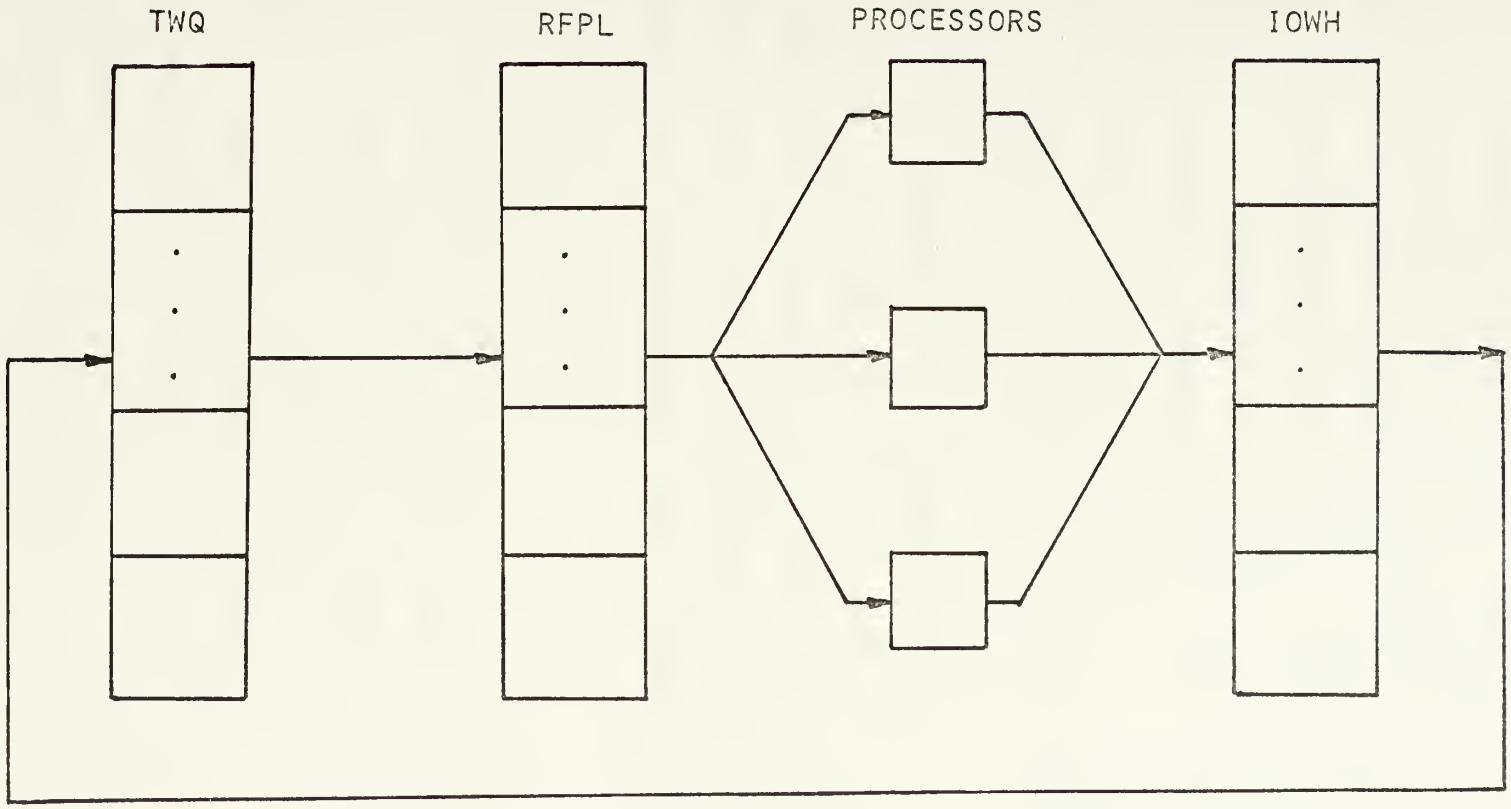

FIGURE 1.

JOB FLOW IN THE LOGICAL MODEL

be general enough that it is possible to add to the structure in any direction and thus to illuminate its interfaces with other viewpoints. Certainly the purposed logical structure loses utility if contradictions arise when such extension is attempted. The central idea, however, is that the difficulties in contemplating complex operating systems (compounded by implementation considerations which properly belong elsewhere) warrant the search for a logical system structure which helps focus attention on relevant system features. The following paragraphs describe a particular logical viewpoint of operating systems which is convenient for discussion of the scheduling problem. The model is suggested by the discussion of multiprocessing schedulers in Lampson [8] and Alexander [2]; it consists of three major or top-level entities: the Ready for Processing List (RFPL), the I/O Wait Heap (IOWH) and the Timer Wait Queue (TWQ). The flow of jobs through the model is shown in Fig 1. No assumptions are made on the characteristics of the processors. Some simple extensions to this basic system will be suggested.

B. READY FOR PROCESSING LIST

The focal point of the model is the Ready for Processing List (RFPL), the queue from which a job is assigned to a processor. The job at the top of the RFPL is assumed to be that job which logically should be the next job to get a processor. This assumption is made in order to sidestep questions of priority, preemption and other details of queue management. Of course, these subtopics are of great interest in the overall scheduling problem since they are the domain of specific implementations of various scheduling algorithms. These details can be considered separately at a lower level without unnecessarily

cluttering the system overview. The point is that this logical division of the problem is natural and convenient in the sense that it excludes consideration of implementation difficulties when attempting to deal with problems about overall system organization. On the other hand, the structure lends to considerations of implementation details a sense of perspective on the larger scale of the scheduling problem.

With this concept of the RFPL in mind, one can view the processor scheduling algorith in the following way: When a processor becomes idle, the top element of the RFPL is inspected. If this element is null then the processor remains in the idle state. If not, the free processor is assigned as the executor for the top job. For convenience it can be assumed that this assignment process includes applicable modifications to the RFPL. Assumed also is a mechanism which detects the addition of a job to an empty RFPL and rechecks for idle processors. Notice again the rather clean logical division of concepts that goes on here: considerations such as time-slicing and priority are relegated to a lower stratum in the hierarchy for separate treatment. Increasing the number of processors in the system similarly adds no complications to the model.

An example of an extension to the logical processor scheduler is a preemption mechanism. To add this capability, we include in the RFPL management routine a section of code which works for the scheduler. This segment compares the logical priority (a term intended to suggest a priority level computed by an arbitrarity complex but lower-level routine) of each new job arriving at the RFPL with that of each job currently running on a CPU. Any time the arriving priority is greater

than some current priority, a job switch will occur. Because of the way the system has been defined, this switch can also be looked at rather simply as the exchange of a job on some processor with the job at the top of the RFPL. The possibility of some more complicated insertion into the RFPL need not be considered at this level.

C. I/O WAIT HEAP

The IOWH holds task control blocks (TCBs) while the corresponding jobs are in a blocked state awaiting completion of I/O operations. Details of queue organization are not restricted in any way at this level. The operation of the IOWH may be viewed in this way: When a task requests an I/O operation, it is blocked and moved from its processor to the IOWH. When the interrupt signaling completion of its I/O occurs, the interrupt handler notifies the IOWH list manager of this fact; the latter routine then retrieves the indicated control block and the job is ready to continue through the system. The exact timing of the above process, and the specific assignment of administrative functions to routines, is purposely vague since these implementation details are not considered at the top level.

D. TIMER WAIT QUEUE

The timer wait queue is used by both task processes and system routines. Its components are a queue and a timer. The enqueued blocks contain pointers to tasks and times associated with each task. The user tasks on this list can be in any one of three states (blocked, ready, running). TWQ entries are assumed to be ordered chronologically by time. The time field nearest the present (real) time is at the head of the list. When the timer interrupts the system, the top TWQ

block is removed from the TWQ. The difference between current time and the time in the next TWQ block is computed and the timer is reset. A variety of things may happen to a job that is removed from the TWQ; for example, it may be run on a preemptive basis or it may simply be made ready and placed on the RFPL. Note that it is sufficient (though possibly awkward) to consider that all tasks enqueued on the TWQ execute preemptively; in this case they behave like one-of-a-kind timer interrupt handlers which can cause the occurrence of any event the modeler desires. The job flow of Fig. 1 shows all jobs entering the RFPL via the TWQ. This could be implemented by assigning a job which desires a delay of zero a TWQ block containing the current time.

The TWQ adds the flexibility to the model necessary for the interpretation of more complex systems. For example, suppose that it is desired to initiate Job X at time T. Assume that it can be initiated as early as time T'=T-t and that there is a heavy penalty for initiating X later than T. To model this situation, a control routine places an artificial process Y on the TWQ which will be run preemptively at time T'. Execution of Y causes X to be unblocked and transferred (possibly from the IOWH) to the RFPL. A new block Y' is then formed on the TWQ which will execute periodically (by rescheduling itself on the TWQ at each execution) until some time just prior to T. The main function of Job Y' is to increase the priority of Job X so that is is maximized just before time T. Thus the probability that Job X will execute before time T can be adjusted with the maximum value to which Y' is allowed to raise the priority of X. Job Y' can thus be thought of as a "watchdog" process. Several of the system procedures described in the program of Part IV use the TWQ in this way.

E. EXTENSIONS

The model described above provides a useful structure in which to build a fairly wide range of schedulers. By logically factoring the system, the model provides a convenient way to assign costs of various kinds to sections of the system being modeled. For example, investigation of system overheads associated with transferring a job from the IOWH to the TWQ is assisted by the model because the source of all such overheads is pinpointed by the structure. In addition, such features as interprocess communication, protection mechanisms and deadlock prevention algorithms can be added to the model without alteration to the basic routines.

Suppose it is desirable to study interdependent processes so that capability for interprocess communication is needed. Then Hansen's scheme [7] for multiple message buffers might be implemented by adding a new entity, the Message Buffer Pool (MBP) to the model. The MBP conceptually contains some finite number of fixed-length message buffers. When process A desires to communicate with process B, MBP is called. If there is an available buffer, it is tagged with the names A and B and loaded with A's message. A true is returned to A. If no buffers are available, false is returned. The A-B buffer is transmitted to the second new entity, the Received Message Array (RMA). The RMA maintains an active entry for each job in the system. Each entry consists of a number of status bits and a pointer to the queue of messages waiting for the corresponding job. When A's message is received at the RMA, B's entry is checked. If B exists, bits in its RMA entry are altered to reflect a non-empty message queue. The buffer automatically remains alloted to A-B until B replies to its message.

When A reads the reply the transaction is considered complete and the buffer is returned to the MBP. If a process sends a message to a non-existant process the system will be responsible for generating a dummy reply for the sender. Obviously, many of Hansen's fine points have been omitted here; the point is that this extension to the basic structure can be made without interfacing difficulties.

This section has been devoted to a discussion of a logical model that seems useful for studying many basic aspects of multiprocessing systems. In Part IV, a computer simulation model of the logical structure described above is presented. Chapter IV, Section B, discusses in detail the implementation of the model entities.

III. REAL-TIME CONSIDERATIONS

The next task is the consideration of the real-time domain. The goals of the following paragraphs are to (1) outline some properties of real-time systems; (2) consider in more detail the characteristics of a real-time system which typifies the class of interest here; and (3) declare some of the specific characteristics and assumptions which underlie the operation of the simulator.

A. PROPERTIES OF THE REAL-TIME ENVIRONMENT

All real-time applications are characterized by a sensitivity to the response time of the system. Modern systems which carry the label "real-time" vary in response times from fractions of hours to milliseconds. Within these loose limits a great variety of application classes have arisen including systems for terminal support, data acquisition, command and control, remote batch, processing control, etc. There seems to be no precise concept or generally accepted definition of "real-time". Rather than propose yet another definition, let us examine some of the attributes of such systems insofar as they relate to the functions of the operating systems of interest in this paper.

A real-time system has the property that at least one process in its job stream is highly time-dependent. Usually, the job stream will contain a mixture of time-critical and non-time-critical jobs. Response times might be expected to lie between five seconds for a terminal system interacting with humans to milliseconds for a process-control system in a petroleum refinery. Such systems commonly deal

with fairly complicated multiple interlocking timing constraints. Thus in any unit time interval the operating system scheduling algorithms must decide the execution order of several processes with possibly interdependent timing requirements.

Peripheral I/O devices not ordinarily associated with a scientific- or business-oriented batch system are often found in real-time systems. This characteristic does not apply so much to time-sharing terminal systems as to process control and command-and-control operations. In general, however, a real-time system will be expected to handle I/O streams which have unorthodox rates and record lengths. Input may arrive at fixed intervals or at variable intervals; it may be supplied to the system on demand or it may interrupt the system. Output may be required to follow a certain input at a certain interval. I/O may occur for some jobs in the stream at a fixed frequency. The information may be transmitted as a single message or as a burst of messages. I/O record lengths may follow a known statistical distribution, they may be predictable from current data, or upper and lower limits may be known. These properties are significant to the operating system because they may define bounds on execution times for the various tasks. For example, the amount of data editing required will be closely related to the number of different record lengths that must be taken into a account. This editing represents a lower bound on processing time for the task procedures since the input data will not be recognized by the task procedures until they are edited. On the other hand, the rate of fixed-frequency arrivals may impose upper bounds on processor time per task if processing overlap is to be avoided.

The tasks executed by a real-time system are sometimes catagorized by the response times they require of the system. The most time-critical jobs in a representative real-time system are constrained to response times of small fractions of a second with very small tolerance. A second job class would require response in seconds to minutes and the precision of these limits might to be specified by some distribution function. Finally, a typical real-time system processes jobs with delayed response requirements measured in minutes to hours.

B. GUN FIRE CONTROL SYSTEM DESCRIPTION

To continue the investigation of real-time system properties, consider a particular system embodying a variety of real-time mechanisms. Such a system is the MK 86 MOD 2 Gun Fire Control System (GFCS) employed on some Naval vessels. The characteristics of the MK 86 GCFS are taken from Ref. [11] which is a study done for the Naval Ordnance Systems Command. The primary purpose of the study was to define requirements for the operating system of the AN/UYK-7 computer which forms the core of the GFCS.

The study defines three process types which differ in the timing constraints they place on the system. Fixed-interval processes require a certain precise time interval between initiations. This implies that the operating system scheduler have the ability to preempt processes in order to supply fixed-interval jobs with a processor when needed. Fixed-rate processes must be initiated at a fixed frequency. Within some time interval such jobs may be initiated at any point so long as the required number of initiations have occured at the end of the interval. Any slippage in a computed initiation time for a fixed-rate

job must be accounted for in the timing of the next initiation in order to maintain overall system timing. Cyclic processes must be completed a fixed number of times per second. The time between successive initiations is not a consideration. However, once a cyclic process is initiated it must run to completion. Therefore the operating system must determine the amount of time available to execute a cyclic process; if there is insufficient time to complete the task, it must not be initiated. Finally there are the asynchronous jobs which usually require immediate handling to prevent loss of data. These jobs typically preempt any other process.

> "The primary mission of a Gun Fire Control System is the tracking of air, surface, and shore targets, and the directing and firing of the ships guns at designated targets. In this capacity, the system computer program must receive target data, perform target tracking, compute a ballistic solution, and position and control the assigned guns." Ref. [11], page 3-1.

Execution of this mission requires connection of the computer to a number of peripheral devices whose diverse constraints are a strong force in the shaping of the executive routine. The following listing contains the I/O requirements for each class of device.

-Track-while-scan (TWS) radar: Data are transmitted to the computer each time a target is detected. This input is variable interval type whose interval length depends upon the number of targets, target movement etc. Data must go out to the TWS radar at a fixed time interval after each TWS input.

-Air tracking radar: Input is fixed-rate, output is fixed-rate at the same rate.

-Remote Optical Sights: Input at fixed rate, output fixed-rate.

-Target Data Transmitter: Input at a fixed rate; no output.

-Consoles and displays: Fixed-rate input and output; rates vary among the different devices.

-Ship's systems: Fixed-rate input; no output.

-Ship's command and control system: Fixed-rate input and output.

-Other ordnance systems: Fixed-rate input and output.

-Gun servo systems: Fixed-rate input and output.

The purpose in the above listing is to indicate the complexity of the I/O stream. Each of these devices requires a task devoted to accepting its inputs at the proper time, computing required results and perhaps outputting data on a specific schedule. There is also a ballistic solution which must be computed by a successive approximation technique until the predicted impact point converges with predicted target position. This calculation requires inputs from many of the above-listed peripherals. The ballistic solution is essentially calculated as a background job which is multiprogrammed with the I/O device handlers.

Thus the operating system that will supervise these computations is constrained by the timing requirements of three job types: fixed-interval initiation, fixed-rate execution and asynchronous execution of TWS data. Slippage in any of these areas results in intolerable error. The designers of this AN/UYK-7 executive assume that multiprocessing will not be required to support the present GFCS; however, a multiprocessor version of the executive is to be prepared in case of excessive system loading by additional processing requirements.

C. HYPOTHETICAL MODEL FOR THE PROGRAM

The hypothetical real-time system that is simulated by the program is in many ways similar to the MK 86 GFCS. In particular, both systems

contain subsystems which belong to three important classes of real-time applications: Process control, exemplified by the gun servo system task; Command and Control; and Data Acquisition (e.g., SINS monitoring), which is characterized by time-constrained input with no output. The following discussion outlines some of the assumptions made in the design of the simulator; other assumptions will be brought out in Chapter IV.

Three job types are distinguished by the simulator. Fixed-frequency jobs are required to complete execution a certain number of times per second but they may start at any convenient time. The scheduling algorithm therefore initiates this job type immediately upon termination of its previous execution. Although the simulator has multiprocessing capability, no special effort is made to initiate more than one fixed-frequency job at a time. In the context of this paper, fixed-frequency jobs are assumed to be rather short; however, there is nothing in the simulator to prevent the user from specifying longer run times for these jobs. They are assumed by the simulator to complete their processor period, perform a single I/O operation (which may involve an arbitrary number of records) and then to terminate. Fixed-frequency jobs may not interrupt other jobs.

Fixed-interval jobs are also conceptually short jobs which execute a single I/O operation after completing all other processing and then terminate. They must initiate at certain fixed times during any unit interval. They are considered the highest-priority jobs and they normally enter the system via an interrupt. When interrupts are not masked off, fixed-interval jobs will always be given a processor on time.

Background jobs are assumed to require somewhat more time and more I/O operations than other jobs. Again, this distinction is conceptual, and the user may specify any run times he desires. Background jobs are assumed to execute for some time interval E, perform an I/O operation, and then repeat this cycle until their total CPU time requirement and total I/O demand are satisfied. The program assumes an excess of main storage so that allocation of core never delays a job; this is a realistic approximation for a system in which all tasks are permanently resident in main storage. However, the skeleton of a core allocation routine is built into the program in the event the user desires to consider core allocation in the overall timing of the system.

The simulator provides various measures with which to evaluate the performance of a particular configuration. The gages actually chosen as significant will depend upon the nature of the experiment. For example, a sample run from an experiment investigating the parameter space of the NAVORD AN/UYK-7 executive discussed above is found in Appendix E. That experiment assumes that fixed-interval jobs always start on time. This condition is forced in the simulator by enabling interrupts. The simulator attempts to initiate a fixed-frequency job as soon as the previous one terminates; initiation will be successful except when all processors are busy. In the latter event the fixed-frequency job is placed on the RFPL to wait. If the system is loaded beyond some critical point, fewer than the required number of fixed-frequency jobs will finish in a given unit interval. This slippage is taken to be an indicator of how well the configuration is handling the applied load.

IV. THE COMPUTER MODEL

A. OVERVIEW

The program chosen to model the real-time systems of Chapter III is a standard discrete time-step event-driven simulator based on MacDougall's BASYS simulator [1] and coded in Stanford ALGOL-W [2]. It is configured with one to ten central processors and two I/O channels. Each simulator run requires the input of about 70 parameters most of which serve to define the job characteristics (see Appendix D). It consists of about 20 procedures, half of which perform utility functions (I/O, statistics gathering, etc.) for the program and a short mainline program which merely serves to initialize the system and set it in motion. Operational details of each procedure in the program are found in Appendix C.

B. IMPLEMENTATION OF THE LOGICAL MODEL

Procedure RFPQ is intended to simulate the RFPL of Part II. It therefore manages its linked-list queue in such a way that the job which represents the "best" choice to get the next free processor is always at the top of the queue. In the model, the "best" choice is simply the least recent arrival at the queue which has a priority no less than any other job on the queue. There is no preemption in the sense that when a job arrives at the queue, no check is made to determine whether a lower-priority job is currently running on some CPU. Incorporation of this feature would require a more general interrupt mechanism than the one built into the program.

The IOWH of Chapter I is implemented via procedure IOWH. The queue manager is designed to search the queue for a job which requires a particular I/O type. Thus, when I/O Device 3 completes an operation, RELIO directs IOWH to determine whether any other jobs are awaiting Device 3.

The TWQ is not implemented as a single procedure. Instead the Event List serves as the TWQ timer and SCHEDULES calls the appropriate routine to service the "timer interrupt." This means that each new TWQ application to be implemented must make three changes in the code: (1) A set of instructions is added to initiate the new TWQ function. Initiation can be accomplished by placing a block on the Event List which contains the desired time of the first instance of the function. The point in the program at which this initiator segment is inserted depends on the specific function to be implemented. (2) A new procedure is added to the program whose purpose is to carry out the desired function. Note that if the new function is to be executed periodically, the function procedure must also place the block for the next execution on the Event List. (3) The case statement in SCHEDULES is modified to call the new function procedure whenever a corresponding TWQ block reaches the top of the Event List. Section A.6 of Appendix A contains examples of TWQ functions implemented in this way.

C. APPLICATIONS

This section describes two simple experiments using the program. The intent is to give some feel for the type of investigations for which the model is useful.

Suppose the program assumptions discussed above are applicable to a user's system. Then the behavior of the average slippage of

fixed-frequency jobs with changes in fixed-frequency CPU requirements may be of interest. This would be the case if the user were contemplating recoding the fixed-frequency job for greater efficiency in an effort to reduce slippage. Job parameters, time slices and overheads in the model are set to provide the desired loading conditions. A series of computer runs is then made with varying fixed-frequency CPU times. This experiment was run with hypothetical parameters for configurations of 1 CPU and 2 CPUs; each run went for 10 simulated seconds. Default parameters were used wherever possible (see Appendix H); under these conditions the processors were in use over 95% of the time. About 75 jobs were completed by the dual-processor configuration on each run; this number averaged 61 for single processor runs. The average RFPQ length for dual-processor runs was about 6; for single-processor runs about 8. All the trial runs were compute-bound; I/O utilization averaged about 50%. The results are tabulated below.

1 CPU: FFREQ CPUTime (msec)	Avg Slippage (jobs/sec)	2 CPUs: FFREQ CPUTime (msec)	Avg Slippage (jobs/sec)
100	2.9991	100	1.8994
90	2.9991	90	1.8994
80	2.99991	80	1.7995
70	2.8991	70	1.6995
60	2.8991	60	1.4996
50	2.2993	50	1.4996
40	2.2993	40	1.0997
30	2.2993	30	1.0997
20	2.1993	20	0.8997
10	2.0994	10	0.9994

As a test of consistency for the program, the data above were interpolated to produce estimates of the fixed-frequency CPU time that would be required to yield an average slippage of 2.0 jobs/sec. for single and dual processor configurations respectively. These estimates were input to the program as data; the resulting slippages (jobs per second) were (single processor) 2.0994 and (dual processor) 1.9994.

Suppose now that under the assumptions above the user desires to study fixed-interval slippage as a function of the number of processors. Since fixed-interval jobs receive considerable special attention in normal operation, simplification of the system would result if such special routines could be dispensed with. This might be possible if a sufficient number of processors were available so that the slippage level is acceptable. Thus a series of computer runs is made with the heavy CPU load imposed by the default parameters. Interrupts are turned off so that fixed-interval jobs receive no special treatment. Again, runs of 10 simulated seconds were made. The CPU requirement for both fixed-frequency and fixed-interval jobs was 50 msec. Between 67 and 81 jobs were completed by the various configurations. CPU utilization ran from 59% (5 CPUs) to 96% (1 CPU) while I/O utilization ranged from 95% (5 CPUs) to 38% (1 CPU). The average FINT delay in the table below is the average time that a FINT job spent on the RFPQ. The results for five configurations are shown below:

No. of CPUs	FINT Slippage (jobs/sec)	Avg FINT delay (msec)
1	3.8	13.50
2	3.1	5.242
3	2.7	1.383
4	1.5	1.105
5	0.5	0.414

Each run of the program requires about 70 input parameters. Of these 70, perhaps a dozen or more would be selected as significant by a particular user. Exploration of the parameter space of these variables is a non-trivial task for which the simulator is useful. Investigation of the transient system state which occurs when the system is first started may also be facilitated by the program. A related area is the definition of the steady-state condition. Effects of various priority structures could be measured. The simulator would be useful for measurements and predictions of various sorts involving system overheads. The major assumption underlying all these suggestions is, as with any simulation model, that the real system can be described by the program.

V. CONCLUSIONS

A logical model with rather general applicability to the study of real-time operating systems has been presented. It has been shown that the constraints which were outlined early in Chapter II are satisfied by the RFPL-IOWH-TWQ model. The logical entities have been translated into a computer program which seems to be a useful tool for investigation of systems whose assumptions are consistent with those of the model.

Various applications were suggested for the program and some simple experiments were run to illustrate conclusions to which the program can lead. For example, from the data tabulated in Chapter IV, Section C, it is seen that fixed-frequency CPU time requirements are able to influence the average fixed-frequency slippage only very slightly. Other parameters in the model seem likely to have a far stronger influence. Also, the data indicate that no value of fixed-frequency CPU time can bring the average slippage to zero in either configuration. The data points are grouped in an interesting fashion, with relatively large values of slippage separating the groups. One is tempted to explain this situation by supposing that the cluster points represent quantum loading levels to which the system is sensitive, but more data are needed to verify this idea. For the dual-processor configuration, the average slippage is monotone non-decreasing at all points except in the region of small CPU time; this minimum value, if it appears in other tests, would be an interesting area for further study.

The data from the second example experiment suggest that increasing the number of processors will not be an economical way to avoid special

routines for handling the fixed-interval jobs. The detailed program output indicates that once these jobs miss a scheduled execution (which happens very quickly) their timing is severely preturbed for the rest of the run; at least half a dozen processors are necessary under the loading conditions of the test in order to insure a timely processor for every fixed-interval job initiation.

Some modifications to the code would produce unqualified improvement. The most urgently needed change is the modification of the normal variate generator in RANDOM to produce statistically acceptable variates (c.f. Appendix F). Algorithm TR from Ref. [1] is recommended. Another change that would simplify the coding of JOBSTART is the modification of RANDOM to return integers instead of reals along with the addition of a procedure to produce real Uniform (0,1) variates. This new routine would be called by RANDOM and GETCPU. The lack of multiple random number streams is also a defect. Another program feature which would make a somewhat more general simulator is a generalized interrupt routine. Such a routine should be coded as a separate procedure capable of simulating several classes of interrupts for any calling job.

The utility of other modifications to the program depends on the particular system being modeled. Changes in this category include extension of the I/O procedure to handle more I/O types, inclusion of a core allocation routine, more direct simulation of supervisor activities, modification of STAT to gather some new statistics, alteration of the overhead accounting scheme, etc.

APPENDIX A. PROGRAM DESCRIPTION

This Appendix describes the overall operation of the program and contains several sections devoted to some peculiarities and special properties of the system. Appendix C, the Program Listing, contains comments that elaborate further on the workings of the program.

A.1 JOBS

The purpose of the model is to provide the user a view of the passage of jobs through the system. A job is defined by a table or block with 9 fields containing such characteristics as total CPU time required, total number of I/O requests to be made to the system, priority, etc. Each job is identified by a number, JOBID, which for the first job is 1 and which increases serially for each job thereafter. Thus in the model, creating a job means deciding what each of the job parameters shall be and placing these numbers in a new block. The job's progress through the model is then marked by the movement of a pointer to its job table among the various procedures that comprise the simulator. As this pointer proceeds on its way it may find itself allocated to a CPU, waiting in some queue for service, performing an I/O operation, or exiting the system. The picture of the system presented to the user is made up of queueing statistics and such system statistics as CPU-I/O overlap and CPU utilization.

A.2 EVENTS

The action within the model is a series of events. An event is some milestone in a job's passage through the system. The events which are possible include creation of a new job, allocation and

deallocation of a processor or an I/O device and exit from the system. It is assumed in this model that a job's time in the system is divided into periods of CPU use alternated with periods of I/O operations. The length of each processing interval (and each I/O interval) is a constant and is chosen so that when a job performs the last of its I/O operations it will require exactly one more CPU period to complete its total CPU requirement. Thus, the sequence of events for a particular job is typically a series of processor allocations and deallocations interspersed with I/O device allocations and deallocations. This pattern, however, may be interrupted in various situations to be discussed later. Note that a job is always in one of three states: processing, doing I/O, or transitioning into or out of the system or between the other two states.

Operation of the model hinges on a chronologically ordered linked list called the Event List. Each entry on this list is an event block containing the information necessary to set the next event into motion. This data includes a pointer to the job which is the subject of the event, the type of event to be performed and the global time (GTIME) at which the event is to occur. The SCHEDULER initiates a simulator cycle by removing the top event block (which represents the event due to happen next), updating GTIME to the time in the block and calling the procedure appropriate for the handling of the event type specified in the block. The SCHEDULER passes to the task procedure the job pointer from the event block; the indicated job becomes the calling job. When the task procedure has finished its work it returns to the SCHEDULER, which then initiates the next simulator cycle. This process continues until the SCHEDULER sees that GTIME has exceeded a

parameter, QGTLIM, which the user has set. At this point the final statistics are collected and printed and the run is terminated.

The Event List is managed by procedure EVBOSS. EVBOSS is called by any procedure which desires to place an event block on the List. EVBOSS forms new event blocks from the data passed to it and inserts them into the List according to two criteria: (1) No blocks preceeding a given block may have an execution time later than that of the given block. (2) If several blocks have the same execution time, then they are ordered according to event type. The order is EXIT, RELIO, IO, RELCPU, GETCPU, JOBSTART. The purpose of this second restriction is to avoid such cases as the following: Suppose that calls to GETCPU and RELCPU are scheduled at the same GTIME on a single-processor configuration. If the GETCPU call is executed first, a job will be sent to the RFPL and without the passage of any simulated time it will be removed from the RFPL. Thus condition (2) causes the simulator to take the view that the job attempting to get a processor should have access to any processors being released at the same instant.

A.3 Task Procedures

Task procedures are those procedures which are concerned with moving jobs through the system. They include JOBSTART, GETCPU, RELCPU, IO, RELIO and EXIT. Each task procedure has at least two functions to perform each time it is called. First, the calling job is accorded the basic service suggested by the task procedure's name. For example, GETCPU first attempts to provide a processor for the calling job, while RELIO takes care of releasing the I/O device which has been allocated to the calling job. Next, each task procedure must decide which will be the next event to occur to the calling job and when it

will occur. For example, when RELIO is called with a job which is somewhere in the middle of its trek through the system, RELIO must realize that the next event for this job will be a GETCPU to initiate its next processor cycle. The "decision" is actually completely deterministic: RELCPU calls IO, IO calls RELIO, RELIO calls GETCPU and GETCPU calls either RELCPU or EXIT. The task procedure signals its decision by causing a new event block to be formed and chained in chronological order onto the Event List. This second function is fundamental to the operation of the model since it is the mechanism whereby the Event List is perpetuated. Every task procedure forms one or more new event blocks every time it is called. Note, however, that with the exception of GETCPU when it is handling an interrupt, only SCHEDULES removes blocks from the list. Note also that while a job is within its CPU period the pointer to its table is located on the Event List in a block requesting a RELCPU. In general, the only time a job table pointer leaves the Event List is when its job is in the transition state. The most important exception to this rule occurs when the job is awaiting some service on one of the queues. In this case its pointer is kept in a block on that queue.

A.4 JOB INITIATION

Jobs enter the system through JOBSTART. When JOBSTART is called it is the calling job that is about to enter the system. Its job table has been filled in previously by JOBSTART and JOBSTART will simply pass its pointer on to GETCPU. However, before doing this, JOBSTART constructs a job table for the next job that will enter the system by drawing from various user-specificed distributions. JOBSTART computes

the global arrival time for the new job (again by drawing) and causes an event block containing this time and a pointer to the new job table to be chained onto the Event List. The event type specified in this block is a pointer back to JOBSTART. In effect, then, when the interarrival time has elapsed this new job will be a calling job for JOBSTART and another event block for the succeeding job will be generated.

A.5 SPECIAL-PURPOSE JOBS

There are two classes of jobs in the simulator that are of special interest. These jobs are intended to represent the behavior of the fixed-frequency and the fixed interval jobs of the NAVORD Executive and other real-time systems. Fixed interval (FINT) jobs are those which must be initiated at fixed times during the run. They enter the system via an interrupt assumed to come from a watchdog process on the TWQ. Anytime the interrupt is enabled the FINT jobs will in fact be started on time. The job parameters for FINT jobs are specified by the user and a special section of JOBSTART handles the creation of these jobs. It is assumed that FINT jobs perform a single I/O operation on each execution. FINT jobs are for the purpose of the sample simulation (c.f. Chapter IV, Section C) initiated to handle I/O requests from some external devices; therefore they must run to completion upon each initiation. Anytime a FINT job gets enqueued for a processor (placed on the RFPQ) it will not initiate on time or it has been interrupted and will therefore not complete on time. Thus any time the processor queue RFPQ is called with a FINT job, this slippage is recorded for later printout by the program.

Fixed-frequency (FFREQ) jobs are jobs which must complete at a certain frequency, but which may be initiated at any time convenient to the system. In the model, the user specifies the required completion frequency (QFF). The system initiates one of these jobs 2 milliseconds after the start of each run; thereafter, each time EXIT is called with a FFREQ JOBID it introduces the next one (through JOBSTART) immediately. Thus, with no other jobs in the system, and assuming sufficient available processor time, a run would consist of exactly QFF FFREQ jobs each second for the entire run period. In the usual case, however, FFREQ jobs will not be able to complete at the desired frequency since they can be interrupted but cannot cause interrupts in order to recover a CPU. At the end of each 1-second interval, procedure SPEC determines the difference between the required number and the actual number of completions. The user will normally give these jobs as high a priority as possible in order to ensure that they are not preempted in the processor queue by less important background jobs.

A.6 INTERRUPTS

Whenever GETCPU is called with a FINT job and interrupts are enabled, an interrupt occurs. The interrupt system which is then activated is intended to model in some ways the interrupt system of a multi-processor AN/UYK-7 configuration. Toward this end, the processors in the system are divided into two sets each time an interrupt occurs: those which are free and those which are busy. If either of the sets is empty, then the processor that will handle the interrupt is chosen equiprobably from the other set. If neither set is empty, then it is 10 times more likely that a selection will be made

from the set of free processors. Although the factor 10 was arbitrarily chosen, it is intended to model the fact that during certain portions of an AN/UYK-7 instruction cycle, the processor is sensitive to certain interrupt types. A specific probability is difficult to compute since it represents a very gross assumption about AN/UYK-7 interrupts. In any event the processor to handle the interrupt is selected with equal probability from whichever set is chosen. If a free processor is chosen then it is simply allocated to the interrupt. If a busy processor is chosen then extra steps must be taken to ensure that the interrupted job will be handled logically. First the time remaining in the interrupted job's current CPU period is computed and stored in a field in the job's table. The job is sent to the RFPQ to await a processor. When the job again gets a CPU it will run out this old period before resuming its normal schedule. Finally, there is a block on the Event List which must be removed; it is the block which contains the call to RELCPU that would have terminated the interrupted job's processor period had it been allowed to continue.

It is logically convenient to consider that interrupts occur at the end of each I/O period. Such interrupts take the form of a call by SCHEDULER to RELIO in response to an Event block created by procedure IO. RELIO then schedules the calling job for GETCPU, but only after a user-specified delay (QIOSTOP) which represents the interrupt processing and any other overheads the user wishes to associate with this operation. Interrupts of the type that occur to FINT jobs, e.g., TWQ watchdog process interrupts, are also used directly by routines that operate as part of the simulator supervisor system. For example,

procedure SPEC is designed to check on the actual completion rate of FFREQ jobs. Therefore, an event block whose execution time is GTIME=1 second is placed on the Event List by the Mainline routine which initializes the List. When this block reaches the top of the List a logical interrupt occurs and SPEC is called. After completing its calculations SPEC creates a new event block with an execution time of GTIME+1 second and blocks itself. Similarly the IOCS routine, which controls the output from the program, is driven by TWQ interrupts.

This interrupt facility is very simple and thus quite limited in usefulness in sections of the program other than the FINT job mechanism. The fact that a more general interrupt handler is not available is clearly a weak point in the program; for example, a preemptive RFPQ would require the same type of service that FINT jobs receive but such service is unavailable in the present implementation.

A.7 OVERHEAD ACCOUNTING

System overheads can be inserted into the model by delaying the passage of a job from one event to the next or from a queue to its next event. For example, suppose it were desired to include in the simulation the overhead associated with transferring a job from the RFPQ to a processor. It would be necessary to modify the code in every place where such a transfer occurs (there are two: one in RELCPU and one in EXIT so that a new event block would be formed. This block would specify a call to GETCPU at a global time of GTIME plus the amount of the overhead. Such overheads are built into the model rather arbitrarily to allow for setting up I/O operations and for continuing normal operations after an I/O operation. Users of the simulator will doubtless want to modify its overhead accounting to suit their particular needs.

A.8 I/O IN THE SIMULATOR

The simulator contains two I/O "devices" and two dummy devices which use the parameters furnished for the other two. A "device" is one of four sections of code in procedure IO. Each time IO is called the IOType is retrieved from the calling job's table and used to decide which user-specified distribution parameters will be used to determine the duration of the I/O operation. An event block is then formed whose event type is RELIO and whose time is GTIME plus the interval computed as above. The dummy "devices" were included to simplify the task of adding new I/O devices to the system.

A.9 SIMULATED TIME

The basic time unit in the simulator is assumed to be 1 millisecond. It is toward this basic unit that all printout messages and conversions are oriented. Thus procedure STAT contains the constants appropriate for converting to minutes from milliseconds. There is, however, nothing to prevent the user from considering any other basic unit he desired except the text of the output segments.

In the sample experiments run for Chapter IV, Section C, time was compressed by a factor of about 5 when the simulator was configured with fewer than four processors. Thus a run of 20 simulated seconds required about 4 seconds of execution time on an IBM 360/67 running under OS/MVT.

A.10 COLLECTION OF STATISTICS

Procedure STAT collects and prints all statistics for the simulator. It is always automatically called at the end of a run. The user may cause a call to STAT to occur at any time(s) during the run by

inserting an appropriate control card, Card 22 (see Appendix D), into the program input stream. The numbers on this card are considered by the reader procedure to be global times and an event block containing a call to STAT is formed that will execute at each of these times. Appendix G describes the manner in which STAT calculates its outputs.

A.11 INPUT PARAMETERS

Each run of the simulator requires approximately 70 constants to define the jobs and configuration for the run. All of these parameters, with two exceptions, have default values assigned (see Appendix H) and need appear on input cards only to override the defaults. On every run, two cards must appear in the input stream: Card 23, the Output Control Card, and Card 24, the Run Termination Card (see Appendix D for details on each card type.) The Run Termination Card must always be the last card in the input set and must always be immediately preceeded by an Output Control Card. All other cards, if any, may appear in any order in the input stream. The program is designed to execute several input sets in succession and it is these latter two cards that delimit the various input data sets. With one exception, all input cards contain a series of integers. The exception occurs when the user desires to specify an empirical distribution for some job parameter; in this case he will provide a follower card containing real numbers which describe the distribution. In either case the numbers are punched in order and separated by one or more blanks. Appendix D contains a detailed discussion of the format of each input card along with an explanation of the parameters.

The output segments mentioned in connection with Card 23 will print only if the I/O flag, IOF, is on; the numbers on this card

control the flag. The integers 0 and 1 have special meaning not attached to any other integers. A zero tells the output control procedure IOCS that no additional flagged output should be printed; a 1 implies that all flagged output should be printed at the level specified by QIOF. IOCS will not attempt to read any additional output control numbers from Card 23 after seeing either of these special characters. Every output control card must end with either a 0 or a 1. If the first datum on Card 23 is not a 0 or a 1, then it will be interpreted by IOCS as the global time at which the user desires the printing of flagged output at the indicated level to commence. This integer must be followed by another, larger integer which represents the stop time for flagged output. This stop time may now be followed by a zero, indicating no more flagged output is desired, a one, indicating that all flagged output should be printed from this time on, or another start-stop pair. Note that a sequence of start-stop times must be strictly increasing.

Although the data for the output of MAP (see Section A.13) is constantly accumulated by that procedure no matter what the setting of IOF, the MAP output is printed only after a full line (650 msec.) is accumulated. This occurs at GTIMEs that are integer multiples of 650 milliseconds. Only at these times is IOF checked. Therefore, the user must insure through his Card 23 that IOF is equal to 1 at the end of each 650 msec. interval for which MAP output is desired. For example, the sequence (100,645,0) on a Card 23 will be interpreted as follows: At GTIME=100 msec. set IOF=1 (commence printing all flagged output). At GTIME=645 msec. set IOF=0 (stop printing flagged output). Leave IOF set to zero for the remainder of the run (print no more flagged output).

Note that at time 650, when MAP is prepared to print the first chart, IOF=0; therefore MAP discards this data and commences collection of data for the 650-1300 millisecond chart.

A.12 RANDOM NUMBERS

Many cards in the input stream contain parameters that will eventually be used by JOBSTART in calls to RANDOM. Procedure RANDOM returns variates from uniform, exponential and normal distributions of arbitrary parameters (see Appendix F) and variates from a user-specified empirical distribution. It may also simply return a user specified constant. It returns real numbers, never integers. RANDOM always expects three integer parameters; however, the exponential routine and the constant routine actually use only two. READER takes this into account and the user need specify on his input card only those parameters actually used. In the following parameter format table for each function of RANDOM, EX stands for the distribution mean while SIG is the standard deviation.

Type of Variate	First Parm	Second Parm	Third Parm
Uniform(a,b)	1	a	b
Exponential(EX)	2	EX	Ignored
Normal(EX,SIG)	3	EX	SIG
Empirical	4	NPAIRS*	INTERP*
Constant	5	CONSTANT	Ignored

*See text below

Notice that the first parameter tells RANDOM which distribution is being called upon and the remaining numbers are the parameters of that distribution.

An empirical distribution is stored in an array in the form of a stack of ordered pairs (L,R) of real numbers. Each stack contains up to 10 pairs. Sufficient space has been alloted so that every job parameter required by JOBSTART can come from an empirical distribution if the user desires. Each time a variate from an empirical distribution is desired, RANDOM first obtains a variate v from U(0,1). It then scans down the appropriate stack testing for the condition that v is less than the L part of each pair. When this condition is met, RANDOM has bracketed v with the L parts of two consecutive pairs (this assumes of course that the pairs are strictly ordered on the L parts: see example below.) RANDOM now returns either the R part of the lower-bound pair or a value obtained by linear interpolation between the R parts of the bracketing pairs. The decision is based on the value of the third parameter in the call to RANDOM (INTERP in the table above.) If INTERP is equal to 1 then the uninterpolated value is returned; any other value for INTERP results in an interpolation.

Suppose that it is desired that at some point in a program RANDOM return numbers from the set (4., 3., 2., 1.) with equal probability. This would occur if RANDOM returns 4. when v (a U(0,1) variate) is between 1.0 and .75, a 3. for .75<v<.50, a 2. for .50<v<.25 and a 1. for .25<v<0. The first step in setting up for this task is to read the distribution in to RANDOM. Then any call of form REAL:=RANDOM (Q1,Q2,Q3) with appropriate parameters will produce the desired variate. Every empirical distribution is read in on two cards. Therefore any of Cards 1-12 or 14-21 whose first parameter is 4 must be followed by a second card containing in free form the set of pairs for the distribution. Each number in each pair may be either type REAL

or type INTEGER; they will all be converted to type REAL after being read in (recall that RANDOM always returns reals.) The three parameters on the first card of this set are, in order, as follows:

First Parameter - Always the integer 4 for empirical distributions.

Second Parameter (NPAIRS) - An integer between 1 and 10 (inclusive) equal to the number of pairs to be read in for this distrbution. In the example above, NPAIRS=4.

Third Parameter (INTERP) - If INTERP=1 then the linear interpolation will not occur; any other value for INTERP will result in an interpolation. This parameter must always be an integer.

Thus for the example, the first card has the form <s> 4 <sp> 4 <sp> 1 <s> where <s> indicates zero or more spaces and <sp> indicates 1 or more spaces. The "no interpolation" option is arbitrarily specified here. In this case, the second card would be <s> .75 <sp>4.<sp>.5 <sp>3.<sp>.25<sp>2.<sp>0<sp>1.<s>. Note carefully how the intervals specified in the example translate into the numbers on the second input card. In particular, the upper limit for the L parts, 1.0, is assumed, while the lower limit, 0, must be explicitly punched. This assumption does not hold in case interpolation is desired: in this event, both limits and their corresponding R parts must be punched up.

A.13 OUTPUT

Output from the simulator is divided into classes or levels. The user specifies via Card 13 which levels he desires to print on each run. Output at level 0 is considered necessary on every run. Output at level 3, the highest level, is detailed debugging information that the user will generally not want. All levels up to and including the

one specified on Card 13 will be printed. All levels greater than zero produce what is referred to in Section A.11 as flagged output. The user may elect to turn off all output during some time intervals and to print at the specified level during others. This is accomplished with Card 23 as described in Section A.11. The kinds of output included in each level are detailed below. See Appendix E for a sample output.

-Level 0: At the lowest (most important) level is the printout of the input parameters and most of the output from procedure STAT. Each job characteristic name in the parameter output is followed by the parenthesized number of the card on which READER expects to find corresponding parameters for RANDOM. The last segment of parameter output is a listing of those job characteristics for which no card was detected in the input stream; the values printed for these characteristics are default values. The output from STAT is self-explanatory; details of the calculation of each statistic is found in Appendix G.

-Level 1: The only output segments at this level are those generated by procedure MAP. MAP produces a one-dimensional chart depicting the activities within the processors with time. Page width limitations break the chart into 650- millisecond segments which then print down the page. Each segment consists of an axis line, a CPU time-line for each CPU in the system and the segment origin time. The axis line is the time scale line and is divided by 50-millisecond clock marks. The top CPU time-line corresponds to CPU1, the second to CPU2, etc. Each time-line is a dashed line which is interrupted by X's whenever the corresponding CPU is processing a job. Each X represents 5 milliseconds of processor

time. Each series of X's on a time-line begins with the JOBID of the job being run. Thus JOBID 386 running for 20 milliseconds appears on its processor time-line as --386X-- and running for 10 milliseconds as --38--. Any time a job initiates within 5k milliseconds of the end of its segment (where k is the number of digits in its JOBID), none of the JOBID will be printed; in this case the entire job will be marked by X's. Thus JOBID 386 running for 20 msec. but starting at GTIME=641 will appear as --XX in the current segment and XX-- in the following segment. The user must also beware of such situations as ---501XXX--- which might be JOBID 501 running for 30 msec. or JOBID 50612 running for 10 msec. followed by JOBID 1 running for 20 msec. In general the only way to resolve irrational MAP entries (such as JOBID 501 suddenly reappearing long after it should have exited the system) is to examine the Level 3 output to be described below. The segment origin time is printed immediately below the axis line near the left end. If extensive use is to be made of the MAP feature, a different format should be considered, e.g., continuous chart with vertical rather than horizontal orientation.

-Level 2: This level is a dummy in the basic simulator and is included for user convenience in adding additional output types.

-Level 3: This Level produces an output statement each time any task procedure is called. Messages from each call to MAP and from procedure SPEC are also printed at this level. In addition, procedure STAT produces a listing of JOBIDs on the Event List, IOWH and RFPQ which prints at Level 3 each time STAT is called. Each of these messages except the ones from SPEC and STAT begin with the procedure name of the sending procedure. The message from SPEC begins with

"&&& FFREQ SLIP:'. The second field of each Level 3 message with the above exceptions contains the JOBID of the calling job. The information following the JOBID, if any, is listed below:

RFPQ: Priority of calling job.

RELCPU: JOBID of job on CPU 1,...JOBID of job on CPU QCP. Zero indicates a free CPU.

EXIT: Total CPU time used by the calling job.

MAP: Same as RELCPU.

The message from SPEC contains the FFREQ slippage (i.e., the difference between the required number of job completions specified by QFF and the actual number of completions) for the particular 1-second interval indicated in the message.

APPENDIX B. SAMPLE DECK SETUP

The following listing shows the deck setup for running the program on the IBM 360/67 at the W.R. Church Computer Center of the Naval Postgraduate School. The User's Manual published by the Center has additional information and explanation of the JCL.

```
 1. //xxxxxxxx JOB (yyyy,zzzz,aaaa),'bb...b'
 2. //  EXEC  ALGOLW,REGION=130K
 3. //SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=3458,SPACE=(TRK,(5,1))
 4. //ALGOL.SYSIN  DD  *
 5. Algol Source Deck
 6. //GO.SYSPRINT  DD  SYSOUT=A,DCB=BLKSIZE=3458,SPACE=(TRK,(5,1))
 7. //GO.SYSIN DD *
 8. 13  1000  2  50  75  4  3  0  100 Any Comment
 9. 9  5  75 Any Comment
10. 14  1  1  4 Any Comment
11. 23  1 Any Comment
12. 24  0 Any Comment
13. /*
```

The alpha fields on Line 1 are job accounting fields; see the User's Manual. Lines 3 and 6 are optional; they reduce the SYSOUT space requirements of the program. Lines 8 and 9 are optional input cards. Lines 12 and 13 are required on every run. Any other data cards would be inserted between lines 7 and 17.

```
BEGIN
RECORD JOBTAB(INTEGER JOBID,PRIORITY,COR,CPU,IOTYPE,ENTIME,IOREQ,IOIRT,
INT);
RECORD EV (INTEGER E,TIME; REFERENCE(JOBTAB)JCB; REFERENCE(EV) PT);
RECORD CQ (REFERENCE(JOBTAB) JT; REFERENCE(CQ) PBX,PF);
RECORD RFP (REFERENCE(JOBTAB) PTER; REFERENCE(RFP) PRE,FOL);
RECORD IOQ(REFERENCE(JOBTAB) JTAB; REFERENCE(IOQ) XPRE,XFOL);
REFERENCE(RFP) CPUP; REFERENCE(EV) ETCP; REFERENCE(IOQ) ITOP;
INTEGER  P,IOF,AVAILCORE,GTIME,OTIME,IOTIME,L,M,NCORE,ACORE,JOBTCT,ATAT,
CPUTOT,LB,LST,STFLG,IID,IDF,FFCNT,QFF,AC;  INTEGER ARRAY CPASSGN(1::10);
INTEGER RLEN,ILEN,RJOBS,IJOBS,RFJS,ICJS,ISCT,RSCT,FFSLIP,FISLIP,MAXFFSL;
INTEGER ARRAY B(1::10); INTEGER ARRAY F(1::10);
REAL ARRAY QEMP(1::8,1::10,1::2); INTEGER ARRAY QTSL(1::10);
REAL RN; INTEGER ARRAY IFLG(1::4); INTEGER ARRAY CFLG(1::10);
             LOGICAL INTFLG; INTEGER QIOS,QIOSTOP,QGTLIM,QCP,QIOF,QIMSK;
             INTEGER STRCTIM,STRETIM; INTEGER ARRAY CREM(1::10);
STRING(5) ULIN; STRING(131) ARRAY CSTR(1::10); STRING(131) AXIS,BLANK;
INTEGER QIP1,QIP2,QIP3,QIM1,QIM2,QIM3,QIC1,QIC2,QIC3,QII1,QII2,QII3,
QIA1,QIA2,QIA3,QIN1,QIN2,QIN3,QFP1,QFP2,QFP3,QFM1,QFM2,QFM3,QFC1,QFC2,
QFC3,QFI1,QFI2,QFI3,QFA1,QFA2,QFA3,QFN1,QFN2,QFN3;
INTEGER QM1,QM2,QM3,QC1,QC2,QC3,QI1,QI2,QI3,QP1,QP2,QP3,QA1,QA2,QA3,QN1,
QN2,QN3,QIO11,QIO12,QIO13,QIO21,QIO22,QIO23;
REFERENCE(JOBTAB) TEM; INTEGER SIGR,SIGI,BCT,FCT,ICT; REAL SSQR,SSQI;
LOGICAL ARRAY CCFLG(1::23); STRING(7) ARRAY CHTR(1::21); REAL SSQTT;
INTEGER FSLT,MAXEPS;

COMMENT***** THE BASIC TIME INTERVAL WITHIN THE SIMULATOR IS ONE MSEC;
```

```
REAL PROCEDURE RANDOM (INTEGER VALUE FN,S,T);
     BEGIN REAL V; INTEGER JV;
     CASE FN OF BEGIN GOTO UNIF; GOTO EXP; GOTO NOR; GOTO EMP; GOTO CON;
     END;

     UNIF: L:=1049*L REM M;
     V:=ROUNDTOREAL(S+(T-S)*(L/M)); GO TO XITZ;

     EXP: V:=ROUNDTOREAL(-S*LN(RANDOM(1,0,1))); GO TO XITZ;

     NOR:V:=0; FOR I:=1 UNTIL 12 DO V:=V+RANDOM(1,0,1);
     V:=ROUNDTOREAL(T*(V-6.)+S); GO TO XITZ;

     EMP: JV:=1; V:=RANDOM(1,0,1); WHILE V<QEMP(S,JV,1) DO JV:=JV+1;
     IF T=1 THEN V:=QEMP(S,JV,2)+.5 ELSE
     V:=QEMP(S,JV,2)+((V-QEMP(S,JV,1))*(QEMP(S,JV-1,2)-QEMP(S,JV,2)));
     GO TO XITZ;

     CON: V:=S+.5;
     COMMENT* .5 IS ADDED TO INSURE CORRECT CONSTANT AFTER TRUNCATION *;

     XITZ: ;     V
     END;
```

```
PROCEDURE SCHEDULES;
     COMMENT**********************************************************
     * Q HOLDS THE EVENT TYPE FOR THE NEXT EVENT. THE TOP BLOCK ON THE *
     * EVENT LIST IS POINTED TO BY ETOP. PT IS THE POINTER IN EACH     *
     * EVENT BLOCK POINTING TO THE NEXT BLOCK ON THE LIST. ETOP:=PT    *
     * (ETOP) CHAINS A BLOCK OFF THE LIST.                             *
     **********************************************************;
     BEGIN REFERENCE (EV) Q;
          START: Q:=PT(ETOP); GTIME:=TIME(Q);IF GTIME>QGTLIM THEN GOTO
           ENDIT; PT(ETOP):=PT(Q);
           IF GTIME>STRETIM THEN BEGIN MAP(0,0,0,1); STROTIM:=STRETIM;
           STRETIM:=STROTIM+650; END;
           CASE E(Q) OF BEGIN
           JOBSTART(JOB(Q)); CORE(1,JOB(Q)); GETCPU(JOB(Q));
           RELCPU(JOB(Q)); IO(JOB(Q)); RELIO(JOB(Q)); EXIT(JOB(Q));
           SPEC(JOB(Q)); IOCS(JOB(Q)); STAT; SAMPLE(JOB(Q)); END;
     GO TO START;
ENDIT: END;

PROCEDURE EVBOSS (INTEGER ET,T; REFERENCE(JOBTAB) P);
     COMMENT**********************************************************
     * EVBOSS CHAINS EVENT BLOCKS ONTO THE EVENT LIST IN ORDER. THE    *
     * LIST IS INITIALIZED WITH A DUMMY FIRST BLOCK AND A DUMMY LAST   *
     * BLOCK TO FACILITATE CHAINING ON BLOCKS. ETOP ALWAYS POINTS TO   *
     * THE DUMMY FIRST BLOCK. TEMP IS A POINTER USED TO CHAIN DOWN THE *
     * LIST TO THE LOCATION AFTER WHICH A NEW ENTRY WILL BE MADE. THIS *
     * CHAINING IS DONE BY THE FIRST WHILE STMT. SECOND WHILE STMT     *
     * ORDERS BLOCKS CONTAINING THE SAME TIME. PROCEDURE PARAMETER ET  *
     * IS THE EVENT TYPE THAT WILL BE USED IN THE CASE STMT IN         *
     * SCHEDULES. T IS THE GTIME AT WHICH THIS EVENT WILL OCCUR.       *
     **********************************************************;
     BEGIN REFERENCE(EV) TEMP; INTEGER KR;
     TEMP:=ETOP;
     WHILE(T> TIME(PT(TEMP)))AND(TIME(PT(TEMP))>0) DO TEMP:=PT(TEMP);
     KR:=TIME(PT(TEMP)); IF T=KR THEN
     WHILE(TIME(PT(TEMP))=KR)AND(E(PT(TEMP))>ET) DO TEMP:=PT(TEMP);
     PT(TEMP):=EV(ET,T,P,PT(TEMP));
     END;
```

```
PROCEDURE JOBSTART (REFERENCE(JOBTAB) H);
     COMMENT**********************************************************
     * IID IS THE JOBID CNTR FOR FINT, IDF FOR FFREQ & LB FOR BKGND.   *
     * THE CALL TO CORE NEAR THE END PASSES THE CALLING JOB INTO THE   *
     * SYSTEM. THIS EXIT IS TAKEN EVERY TIME EXCEPT ON THE FIRST BKGND *
     * CALL.                                                           *
     **********************************************************;
     BEGIN INTEGER MS,ICTP,CPUT,PRI,NOREQ,II,ARRVT;
     REFERENCE(JOBTAB) TEMP2;
     IF IOF>2 THEN WRITE("                              JOBSTART:",JOBID(H));
     IF JOBID(H)>499  THEN IF JOBID(H)<800   THEN GOTO FQ ELSE GOTO FT;
     COMMENT...CODE FROM HERE TO FT PRODUCES BACKGROUND JOBS;
     MS:=TRUNCATE(RANDOM(QM1,QM2,QM3));
     CPUT:=TRUNCATE(RANDOM(QC1,QC2,QC3));
     ICTP:=TRUNCATE(RANDOM(QI1,QI2,QI3));
     LB:=LB+1; IF LB>499 THEN LB:=1; COMMENT..LB::=JOB ID (BACKGROUND);
     PRI:=TRUNCATE(RANDOM(QP1,QP2,QP3));
     ARRVT:=TRUNCATE(RANDOM(QA1,QA2,QA3));
     ARRVT:=ARRVT+GTIME;
     NOREQ:=TRUNCATE(RANDOM(QN1,QN2,QN3)); IF NOREQ<1 THEN NOREQ:=1;
     COMMENT..VARIATES <1 TRUNCATE TO 0 (ABOVE) AND CAUSE DIVISION BY
     ZERO (BELOW);
     II:=TRUNCATE (CPUT/NOREQ);
     TEMP2:=JOBTAB(LB,PRI,MS,CPUT,IOTP,ARRVT,NOREQ,II,0);
     EVBOSS(1,ARRVT,TEMP2); IF PRIORITY(H)>0 THEN GOTO PASS
     ELSE GO TO ALTO;
     COMMENT..THE ONLY TIME H=0 IS WHEN THE FIRST BKGND JOB ENTERS;

     FT: IID:=IID+1; NOREQ:=TRUNCATE(RANDOM(QIN1,QIN2,QIN3)); CPUT:=
     TRUNCATE(RANDOM(QIC1,QIC2,QIC3));
     TEMP2:=JOBTAB(IID,TRUNCATE(RANDOM(QIP1,QIP2,QIP3)),0,
                   CPUT,TRUNCATE(RANDOM(QII1,QII2,QII3)),GTIME+
     TRUNCATE(RANDOM(QIA1,QIA2,QIA3)),NOREQ,TRUNCATE(CPUT/NOREQ),0);
     EVBOSS(1,ENTIME(TEMP2),TEMP2); GO TO PASS;

     FQ: IDF:=IDF+1; IF IDF>799 THEN IDF:=500;
     NOREQ:=TRUNCATE(RANDOM(QFN1,QFN2,QFN3)); CPUT:=
     TRUNCATE(RANDOM(QFC1,QFC2,QFC3));
     TEMP2:=JOBTAB(IDF,TRUNCATE(RANDOM(QFP1,QFP2,QFP3)),TRUNCATE(RANDOM
     (QFM1,QFM2,QFM3)),CPUT,TRUNCATE(RANDOM(QFI1,QFI2,QFI3)),GTIME,
     NOREQ,TRUNCATE(CPUT/NOREQ),0);
     GETCPU(TEMP2); GO TO ALTO;
     PASS: CORE(0,H);   ALTO: END;

PROCEDURE CORE (INTEGER I; REFERENCE(JOBTAB) JPT);
     BEGIN IF I=0 THEN GETCPU(JPT); END;
```

```
PROCEDURE GETCPU(REFERENCE(JOBTAB) JCBP);
    COMMENT**********************************************************
    * QCP IS THE NUMBER OF CPUS. CFLG(I)=1=>CPU(I) IS BUSY.CPASSGN(I) *
    * HOLDS THE JOBID OF THE JOB RUNNING ON CPU(I). CPASSGN(I)=0 IF   *
    * CPU(I) IS FREE. INTFLG IS SET BY PROCEDURE RTIME. IT INDICATES  *
    * WHETHER THE CPU TIME BEING ASSIGNED IS A NEW SLICE. ONLY THEN   *
    * IS IOREQ DECREASED. INTRPT: S COUNTS BUSY CPUS. CPUS ARE DIVIDED*
    * INTO BUSYS AND FREES BY FIRST IF STMT. RN DECIDES WHETHER FREE  *
    * OR BUSY WILL HANDLE THE INTERRUPT, RNO PICKS A PARTICULAR CPU.  *
    * CP IS ASSIGNED THE NO OF THE CHOSEN CPU. D6: FIRST NINE LINES   *
    * LOCATE INTERRUPTED JOBS BLOCK ON EV LIST & REMOVE IT. INT IS A  *
    * FIELD IN JOB TABLE & HOLDS AMOUNT OF CPU TIME REMAINING AFTER   *
    * INTERRUPT. **********************************************************;
    BEGIN INTEGER IA,RTIM,JI; REFERENCE(JOBTAB) TEM1;REFERENCE(EV) EVP,
    EVQ; INTEGER S,CP,BP,FP,MG; REAL T,RNO; IA:=0; JI:=JOBID(JCBP);
    IF IOF>2 THEN WRITE("                    GETCPU:",JOBID(JCBP),GTIME);
  FOR I:=1 UNTIL QCP DO IF CFLG(I)=0 THEN BEGIN IA:=I; GO TO NEXT; END;
    NEXT: IF (JI>799 )AND(QIMSK=0) THEN GO TO INTRPT;
    IF IA¬=0 THEN BEGIN
    RTIM:=RTIME(JCBP,QTSL(IA),IOIRT(JCBP)); CPUTOT:=CPUTOT+RTIM;
          CFLG(IA):=1; OVRLAP; CPASSGN(IA):=JI;
          IF JI>499  THEN BEGIN EVBOSS(4,GTIME+RTIM,JCBP); GOTO W7; END;
          IF IOREQ(JCBP) =0 THEN EVBOSS(7,GTIME+RTIM,JCBP)
          ELSE BEGIN IF INTFLG THEN IOREQ(JCBP):=IOREQ(JCBP)-1;
          EVBOSS(4,GTIME+RTIM,JCBP); END;
      W7:MAP(JI,RTIM,IA,0); END
    ELSE JCBP:=RFPQ(1,JCBP); GO TO XIT;

  INTRPT: S:=MG:=0; FP:=BP:=1; RN:=RANDOM(1,0,1);IF IA=0 THEN RN:=.01;
    FOR I:=1 UNTIL QCP DO IF CFLG(I)=1 THEN BEGIN B(BP):=I; S:=S+1;
       BP:=BP+1; END ELSE BEGIN F(FP):=I; FP:=FP+1; END;
    RNO:=RANDOM(1,0,1); IF S=0 THEN GO TO D3;
    IF RN<.1 THEN BEGIN T:=1./S;
          FOR IA:=1 UNTIL S DO IF RNO<T*IA THEN BEGIN CP:=B(IA);
          GO TO D6; END; END;
 D3:T:=1./(QCP-S); FOR IA:=1 UNTIL QCP-S DO IF RNO<T*IA THEN BEGIN
          CP:=F(IA); GO TO D6; END;
 D6:IF CPASSGN(CP)¬=0 THEN BEGIN MG:=2; EVP:=PT(ETOP); EVQ:=ETOP;
          WHILE JOBID(JOB(EVP))¬=CPASSGN(CP) DO BEGIN EVQ:=EVP;
          EVP:=PT(EVP); END;
          IF (E(EVP)¬=4)AND(E(EVP)¬=7) THEN WRITE("**********ERROR...
          ASSIGNMENT EXCEPTION IN GETCPU");
    IA:=TIME(EVP)-GTIME; INT(JOB(EVP)):=INT(JOB(EVP))+IA;
    CPUTOT:=CPUTOT-IA; TEM1:=RFPQ(1,JOB(EVP)); PT(EVQ):=PT(EVP); END;
    CPASSGN(CP):=JI; CFLG(CP):=1; OVRLAP;
    RTIM:=RTIME(JCBP,QTSL(CP),IOIRT(JCBP)); EVBOSS(4,GTIME+RTIM,JCBP);
    MAP(JI,RTIM,CP,MG); CPUTOT:=CPUTOT+RTIM;
 XIT: END;
```

```
REFERENCE (JOBTAB) PROCEDURE RFPQ  (INTEGER X; REFERENCE(JOBTAB)JCBP);
     COMMENT**********************************************************
     * RFJS ACCUMULATES JOB-SECONDS FOR RFPQ. RLEN IS THE INSTANTANEOUS*
     * QUEUE LENGTH. RSCT HOLDS THE TIME OF THE LAST QUEUE STATE       *
     * CHANGE. FISLIP COUNTS FINT SLIPS. JCBP¬=NULL=>CHAIN JOB ONTO    *
     * RFPQ, JCBP=NULL=>RETURN TOP POINTER IN QUEUE.                    *
     **********************************************************;

COMMENT***** BIGGER NUMBERS HAVE HIGHER PRI, FIFO WITHIN A PRI;

     BEGIN REFERENCE (RFP)  RA,TEMP3; REFERENCE(JCBTAB) Q1; INTEGER DEL;
     RFJS:=RFJS+RLEN*(GTIME-RSCT); RSCT:=GTIME;
     IF JCBP¬= NULL THEN
          BEGIN
          IF IOF>2 THEN WRITE("                                   RFPQ:",
          JOBID(JCBP),PRIORITY(JCBP));
     IF JOBID(JCBP)>799 THEN BEGIN FISLIP:=FISLIP+1; COR(JCBP):=GTIME;
     END;
          RA:= CPUP; Q1:=NULL; RLEN:=RLEN+1; RJCBS:=RJOBS+1;
          WHILE (FOL(RA)¬=NULL) AND
          (PRIORITY(JCBP)<=PRIORITY(PTER(RA))) DO RA:=FOL(RA);
          TEMP3:=RFP (JCBP,PRE(RA),RA);
          IF PRE(RA)¬=NULL THEN FOL(PRE(RA)):=TEMP3 ELSE CPUP:=TEMP3;
          PRE(RA):=TEMP3;
          END
     ELSE  BEGIN
     IF RLEN>0 THEN RLEN:=RLEN-1;
     IF PTER(CPUP)¬=NULL THEN
          BEGIN Q1:=PTER(CPUP); CPUP:=FOL(CPUP); PRE(CPUP):=NULL;
          IF JOBID(Q1)>799 THEN BEGIN DEL:=GTIME-COR(Q1); FSLT:=FSLT+
          DEL; IF MAXEPS<DEL THEN MAXEPS:=DEL; END; END
          ELSE Q1:=NULL;
          END;
     Q1
END;

PROCEDURE RELCPU (REFERENCE(JOBTAB) JBTB);
     COMMENT* VARIABLES USED HERE ARE DEFINED IN GETCPU COMMENTS;
     BEGIN REFERENCE(JOBTAB) NEW;
     IF IOF>2 THEN BEGIN WRITE("RELCPU:",JOBID(JBTB)); FOR M:=1 UNTIL
     QCP DO WRITEON(CPASSGN(M)); WRITEON(GTIME); END;
     FOR I:=1 UNTIL QCP DO IF CPASSGN(I) =JOBID(JBTB) THEN BEGIN
          CFLG(I):=0; CPASSGN(I):=0; OVRLAP; GO TO W1; END;
     WRITE("*******ERROR....RELCPU CALLED WITH NO BUSY CPU******");
 W1: EVBOSS(5,GTIME+QIOS,JBTB);  COMMENT*QIOS IS ASSUMED C/H FOR I/OSET;
     NEW:=RFPQ(1,NULL); IF NEW¬=NULL THEN GETCPU(NEW); END;
```

```
PROCEDURE IO (REFERENCE(JOBTAB) JOBT);
     COMMENT* IFLG(I)=1=>I/O DEVICE(I) IS BUSY.                          *;
     BEGIN INTEGER T; REFERENCE(JOBTAB) D2;
     IF IOF>2 THEN WRITE("                              IO:",JOBID(JOBT),GTIME);
     IF IFLG(IOTYPE(JOBT))=1 THEN BEGIN D2:=IOWH(1,JOBT); GO TO EXIT;
     END;
     CASE IOTYPE(JOBT) OF BEGIN
          GOTO IOT1; GOTO IOT2; GOTO IOT3; GOTO IOT4; END;
IOT1:T:=TRUNCATE(RANDOM(QIO11,QIO12,QIO13)); IOTIME:=IOTIME+T;
     GO TO EXT1;
IOT2:T:=TRUNCATE(RANDOM(QIO21,QIO22,QIO23)); IOTIME:=IOTIME+T;
     GO TO EXT1;
                         IOT3: GOTO IOT2; IOT4: GOTO IOT1;
   EXT1: IFLG(IOTYPE(JOBT)):=1; OVRLAP; EVBOSS(6,GTIME+T,JOBT);
     EXIT: END;

REFERENCE(JOBTAB) PROCEDURE IOWH (INTEGER TYP; REFERENCE(JOBTAB) JBPT);
     COMMENT**************************************************************
     * IOJS ACCUMULATES JOB-SECONDS FOR THE QUEUE. ILEN IS                *
     * INSTANTANEOUS QUEUE LENGTH. ISCT IS TIME OF LAST QUEUE STATE       *
     * CHANGE. JBPT¬=NULL=>ADD NEW JOB TO QUEUE. JBPT=NULL=>SEARCH        *
     * QUEUE FOR JOB WITH I/O TYPE=TYP.                                   *
     **************************************************************;
     BEGIN REFERENCE(IOQ) TA,TEMP4; REFERENCE(JCBTAB) Q4;
     IOJS:=IOJS+ILEN*(GTIME-ISCT); ISCT:=GTIME;
     Q4:=NULL;
     IF JBPT¬=NULL THEN
          BEGIN
          IF IOF>2 THEN WRITE("                                   IOWH:",
          JOBID(JBPT));
          ILEN:=ILEN+1; IJOBS:=IJOBS+1;
          TEMP4:=IOQ(JBPT,NULL,ITOP); XPRE(ITOP):=TEMP4; ITOP:=TEMP4;
          END
     ELSE
          BEGIN
          TA:=ITOP; IF ILEN>0 THEN ILEN:=ILEN-1;
          WHILE (JTAB(TA)¬=NULL)AND(TYP¬=IOTYPE(JTAB(TA)))
          DO TA:=XFOL(TA);
          IF JTAB(TA)¬=NULL THEN
               BEGIN
               Q4:=JTAB(TA);
               IF XPRE(TA)=NULL THEN BEGIN ITOP:=XFOL(TA);
               XPRE(XFOL(TA)):=NULL; END
               ELSE BEGIN XFOL(XPRE(TA)):=XFOL(TA); XPRE(XFOL(TA)):=
               XPRE(TA); END;
               END;
          END;     Q4
     END;
```

```
PROCEDURE RELIO (REFERENCE(JOBTAB) JB);
     BEGIN REFERENCE(JOBTAB) Z;
     IF IOF>2 THEN WRITE("                              RELIO:",JOBID(JB),GTIME);
     IFLG(IOTYPE(JB)):=0; OVRLAP;
     IF JOBID(JB)>499   THEN EVBOSS(7 ,GTIME+75 ,JB)  ELSE
     EVBOSS(3,GTIME+QIOSTOP, JB); COMMENT*QIOSTOP=I/O TAKEDOWN TIME;
     Z:=IOWH(IOTYPE(JB),NULL);
     IF Z¬=NULL THEN IO(Z);
     END;

PROCEDURE EXIT (REFERENCE(JOBTAB) TAB);
     COMMENT**********************************************************
     * ATAT & SSQTT ARE STATISTICAL ACCUMULATORS. EXIT DETERMINES     *
     * WHETHER QFF FFREQS HAVE CALLED DURING THE CURRENT SECOND. IF YES*
     * SCHEDULE THE NEXT FFREQ AT THE START OF THE NEXT SECOND, ELSE  *
     * SCHEDULE IT IMMEDIATELY. FCT COUNTS FFREQ JOBS,ICT COUNTS FINTS *
     * AND BCT COUNTS BKGND JOBS.                                     *
     **********************************************************;
     BEGIN REFERENCE(JOBTAB) Y; INTEGER N, KD, DELT, ID; REAL R;
     ID:=JOBID(TAB); JOBTOT:=JOBTOT+1;
     IF IOF>2 THEN WRITE("EXIT.. JOBID & CPUTOT ARE:",ID,CPUTOT);
     DELT:=GTIME-ENTIME(TAB); ATAT:=ATAT+DELT; SSQTT:=SSQTT+DELT**2;
     FOR NN:=1 UNTIL QCP DO IF CPASSGN(NN) =ID THEN BEGIN
     N:=NN; GO TO W5; END;
     IF (ID>499)AND(ID<8C0) THEN BEGIN FCT:=FCT+1;
          FFCNT:=FFCNT+1; IF FFCNT<QFF THEN BEGIN JOBSTART(TAB);
          GO TO XT; END;
          R:=GTIME; N:=0;  WHILE R>10 DO BEGIN R:=R/10; N:=N+1; END;
          KD:=TRUNCATE(10**N); N:=TRUNCATE(R); EVBOSS(1,(N+1)*KD,TAB);
     END; IF ID>799 THEN ICT:=ICT+1; GO TO XT;
     W5: CFLG(N):=0; OVRLAP; CPASSGN(N):=0; BCT:=BCT+1;
     Y:=RFPQ(1,NULL); IF Y¬=NULL THEN GETCPU(Y);
     CORE(1,TAB);
  XT:END;

PROCEDURE OVRLAP ;
     COMMENT**********************************************************
     * STFLG=1=>SYSTEM IS NOW IN OVERLAPPED STATE. STFLG=0=>NO OVERLAP *
     * EXISTS.                                                        *
     **********************************************************;
     BEGIN
     IF IOF>2 THEN WRITE("OVRLAP: ",GTIME,"CFLG",CFLG(1),CFLG(2),"IFLG",
     IFLG(1),IFLG(2),IFLG(3),IFLG(4));
     FOR I:=1 UNTIL QCP DO IF CFLG(I)=1 THEN GO TO P1;
   P2: IF STFLG=0 THEN GO TO QXIT;
     OTIME:=OTIME+GTIME-LST; STFLG:=0; GO TO QXIT;
     P1: FOR I:=1 UNTIL 4 DO IF IFLG(I)=1 THEN GO TO P3;
     GO TO P2;
```

```
P3: IF STFLG=1 THEN GO TO QXIT; LST:=GTIME; STFLG:=1;
QXIT: END;
```

```
PROCEDURE MAP (INTEGER VALUE JID; INTEGER JLEN,CP,F);
     COMMENT**********************************************************
     * THE FIRST HALF OF MAP DECIDES WHERE ON THE APPROPRIATE TIME-    *
     * LINE THE NEW MAP WILL START AND HOW LONG IT IS & DECODES THE    *
     * JOBID & INSERTS IT INTO THE MAP. CREM HOLDS ANY MAP SEGMENTS    *
     * THAT OVERFLOW ONTO THE NEXT 650 MSEC CHART. THE IF STMT AT THE  *
     * BOTTOM OF THE FIRST HALF BLANKS OUT THE MAP SEGMENT REMAINING   *
     * AFTER A JOB IS INTERRUPTED. WRT: WRITE OUT TIME-LINES, AXIS &   *
     * STRING ORIGIN TIME. WA: BLANK OUT NEW TIME-LINES & MAP IN ANY   *
     * TIMES LEFT OVER FROM THE LAST CHART.                            *
     **********************************************************************;

     COMMENT..F=1=>WRITE, F=0=>OVERLAY A JOB;

     BEGIN INTEGER START,ORG,LIM,IG,B,TG; STRING(4)STRID; REAL IR;
IF IOF>2 THEN BEGIN WRITE("                                    MAP: ",JID); FOR M:=1
UNTIL QCP DO WRITEON(CPASSGN(M)); END;
     IF F=1 THEN GO TO WRT;
     IR:=JID; IG:=0; START:=GTIME-STROTIM;
     IF START>650 THEN BEGIN STROTIM:=TRUNCATE(START/650)*650;
     START:=START-STROTIM; END;
     ORG:=TRUNCATE(START/5); LIM:=TRUNCATE((START+JLEN)/5)-1;
     WHILE IR>10 DO BEGIN IR:=IR/10.; IG:=IG+1; END;
     FOR IB:=IG STEP -1 UNTIL 0 DO BEGIN B:=TRUNCATE(10**IB);
     TG:=TRUNCATE(JID/B); CSTR(CP)(ORG|1):=CODE(TG+240);
     JID:=JID-TG*B; IF ORG<130 THEN ORG:=ORG+1 ELSE GO TO C1; END;
 C1: IF LIM>130 THEN BEGIN TG:=130; CREM(CP):=LIM-130; END ELSE TG:=LIM;
     FOR I:=ORG UNTIL TG DO CSTR(CP)(I|1):="X";
     IF (F=2) AND (TG<130) THEN FOR J:=TG+1 UNTIL 130 DO CSTR(CP)(J|1)
     :=" ";
     GO TO W6;

  WRT: IF IOF=0 THEN GOTO WA; WRITE(" "); WRITE(" ");
     FOR I:=1 UNTIL QCP DO BEGIN WRITE(" "); WRITE(CSTR(I)); END;
     WRITE(AXIS); INTFIELDSIZE:=6; WRITE(STROTIM); INTFIELDSIZE:=14;
     WRITE(" "); WRITE("**********************************");
  WA:FOR K:=1 UNTIL QCP DO BEGIN
     CSTR(K):=BLANK;
     IF CREM(K)¬=0 THEN BEGIN IF CREM(K)<131 THEN BEGIN B:=CREM(K);
     CREM(K):=0; END  ELSE BEGIN B:=130; CREM(K):=CREM(K)-130; END;
     FOR KB:=1 UNTIL B DO CSTR(K)(KB|1):="X";  END; END;
     W6: END;
```

```
PROCEDURE SAMPLE(REFERENCE(JOBTAB) DZ);
     COMMENT*THIS PROCEDURE SAMPLES THE RFPQ & IOWH EVERY 100 MSEC    *;
     BEGIN SIGR:=SIGR+RLEN; SSQR:=SSQR+RLEN**2; SIGI:=SIGI+ILEN;
     SSQI:=SSQI+ILEN**2; EVBOSS(11,GTIME+100,DZ); END;

PROCEDURE IOCS (REFERENCE(JOBTAB) D1);
     COMMENT* THIS PROCEDURE READS CARD TYPE 23 & SETS IOF ACCORDINGLY*;
     BEGIN
     IF P=0 THEN BEGIN IOF:=0; GO TO ZXT; END;
     IF P=1 THEN BEGIN IOF:=QIOF; GO TO ZXT; END;
     IOF:=JOBID(D1); IF JOBID(D1)=QIOF THEN JOBID(D1):=0 ELSE JOBID(D1)
     :=QIOF; EVBOSS(9,P,D1); READON(P); ZXT: END;

INTEGER PROCEDURE RTIME(REFERENCE(JOBTAB)DD; INTEGER VALUE SL,CPER);
     COMMENT**********************************************************
     * RTIME COMPUTES THE CPU PERIOD FOR THE CALLING JOB. INT(DD)=0=>  *
     * START NEW CPU PERIOD, ELSE RUN A SLICE FROM THE OLD PERIOD.     *
     **********************************************************;
     BEGIN INTEGER RTV; INTFLG:=FALSE;
     IF INT(DD)=0 THEN BEGIN INTFLG:=TRUE;
     IF CPER>SL THEN BEGIN RTV:=SL; INT(DD):=CPER-SL; END
     ELSE RTV:=CPER; GO TO D7; END;

     IF INT(DD)>SL THEN BEGIN
     RTV:=SL; INT(DD):=INT(DD)-SL; END ELSE BEGIN RTV:=INT(DD);
     INT(DD):=0; END;  D7: ;    RTV    END;
```

```
PROCEDURE STAT;
      BEGIN INTEGER DEL;
      WRITE("**********S T A T **********************************");
      WRITE(" "); WRITE("******  GLOBAL TIME (MSEC):",GTIME); WRITE(" ");
      WRITE("TOTAL JOBS COMPLETED: ",JOBTOT,"FINT: ",ICT,"FFREQ: ",FCT,
      "BACKGROUND: ",BCT); WRITE(" ");
      WRITE("TOTAL CPU TIME USED, MSEC: ", CPUTOT,"CPU UTILIZATION,%:",
      (TRUNCATE(1000*(CPUTOT/(QCP*GTIME)+.005)))/10);  WRITE(" "); WRITE
      ("TOTAL I/O TIME,MSEC:",IOTIME," I/O UTILIZATION,%:",
      IOTIME/(2*GTIME)); WRITE(" "); WRITE
      ("TOTAL I/O-CPU OVERLAP,MSEC: ", OTIME,"PERCENTAGE OVERLAP: ",
      (TRUNCATE(10000*((OTIME/CPUTOT)+.00005)))/100); WRITE(" ");
      WRITE("AVERAGE TURNAROUND TIME,MSEC: ",TRUNCATE(10*((ATAT/JOBTOT)
      +.05))/10, "   VARIANCE(TURNAROUND)",(JOBTOT*SSQT-ATAT**2)/(JOBTOT
      *(JOBTOT-1))); WRITE(" "); WRITE("THROUGHPUT, JOBS/MIN",(JOBTOT
      *60000)/GTIME); WRITE(" "); WRITE(" ");
      WRITE("TOTAL NO OF FFREQ SLIPS:",FFSLIP,"AVG FFREQ SLIPPAGE/SEC:",
      (FFSLIP*1000)/GTIME,"MAX SLIP:",MAXFFSL); WRITE(" ");
      WRITE("TOTAL NO OF FINT SLIPS:",FISLIP); WRITE(" ");
      IF RJOBS>0 THEN WRITE("AVG FINT SLIP,MSEC:",FSLT/RJOBS,
      "MAX SLIP, MSEC:",MAXEPS); WRITE(" ");
      DEL:=RLEN*(GTIME-RSCT); RFJS:=RFJS+DEL;
      WRITE("***  RFPQ- AVG NO ENQUEUED JOBS: ",TRUNCATE(100*((RFJS/
      GTIME)+.005))/100 );IF RJOBS>1 THEN BEGIN WRITEON("SAMPLE AVG",
      SIGR/RJOBS,"  SAMPLE VARIANCE",
      (RJOBS*SSQR-SIGR**2)/(RJOBS*(RJOBS-1))); WRITE(" ");
      WRITE("AVG WAIT TIME,MSEC: ", ROUND(RFJS/RJOBS)); END; WRITE(" ");
      RFJS:=RFJS-DEL; DEL:=ILEN*(GTIME-ISCT); IOJS:=IOJS+DEL;
      WRITE("***  IOWH- AVG NO ENQUEUED JOBS: ",TRUNCATE(100*((IOJS/
      GTIME)+.005))/100); IF IJOBS>1 THEN BEGIN WRITEON("SAMPLE AVG",
      SIGI/IJOBS,"  SAMPLE VARIANCE",
      (IJOBS*SSQI-SIGI**2)/(IJOBS*(IJOBS-1))); WRITE(" ");
      WRITE("AVG WAIT TIME, MSEC:",ROUND(IOJS/IJOBS)); END; WRITE(" ");
      IOJS:=IOJS-DEL;
      IF IOF<3 THEN GO TO KIO;
      WRITE("******  EVENT LIST AND QUEUE STATUS  ***************");
      WRITE(" ");
WRITE(" "); WRITE("EVENT QUEUE:"); ETOP:=PT(ETOP);
WHILE JOB(ETOP)¬=NULL DO BEGIN WRITE(JOBID(JOB(ETOP)),E(ETOP));
ETOP:=PT(ETOP); END;
WRITE(" "); WRITE("RFPQ:"); WHILE PTER(CPUP)¬=NULL DO BEGIN
WRITE(JOBID(PTER(CPUP))); CPUP:=FOL(CPUP); END; WRITE(" ");
WRITE("IO QUEUE:"); WHILE JTAB(ITOP)¬=NULL DO
BEGIN WRITE(JOBID(JTAB(ITOP))); ITOP:=XFOL(ITOP); END;
 KIO:WRITE(" "); WRITE("**************************************************");
      END;
```

```
PRCCEDURE READER;
     COMMENT**********************************************************
     * CCFLG(I)=1=>CARD TYPE I HAS BEEN READ FRCM INPUT STREAM. USED   *
     * BY DEFAULT DIST FEATURE OF WRITER.                              *
     **********************************************************;
     BEGIN INTEGER CCP,SC;START: READ(CCP); CCFLG(CCP):=FALSE;
     CASE CCP OF BEGIN
        BEGIN READON(QIP1,QIP2); QIP3:=EREAD(QIP1,QIP2); END;
        BEGIN READON(QIM1,QIM2); QIM3:=EREAD(QIM1,QIM2); END;
        BEGIN READON(QIC1,QIC2); QIC3:=EREAD(QIC1,QIC2); END;
        BEGIN READON(QII1,QII2); QII3:=EREAD(QII1,QII2); END;
        BEGIN READON(QIA1,QIA2); QIA3:=EREAD(QIA1,QIA2); END;
        BEGIN READON(QIN1,QIN2); QIN3:=EREAD(QIN1,QIN2); END;
        BEGIN READON(QFP1,QFP2); QFP3:=EREAD(QFP1,QFP2); END;
        BEGIN READON(QFM1,QFM2); QFM3:=EREAD(QFM1,QFM2); END;
        BEGIN READON(QFC1,QFC2); QFC3:=EREAD(QFC1,QFC2); END;
        BEGIN READON(QFI1,QFI2); QFI3:=EREAD(QFI1,QFI2); END;
        BEGIN READON(QFA1,QFA2); QFA3:=EREAD(QFA1,QFA2); END;
        BEGIN READON(QFN1,QFN2); QFN3:=EREAD(QFN1,QFN2); END;
        BEGIN READON(QGTLIM,QCP,QIOS,QIOSTOP,QFF,QIOF,QIMSK); FOR I:=1
                  UNTIL QCP DO READON(QTSL(I)); END;
        BEGIN READON(QP1,QP2); QP3:=EREAD(QP1,QP2); END;
        BEGIN READON(QM1,QM2); QM3:=EREAD(QM1,QM2); END;
        BEGIN READON(QC1,QC2); QC3:=EREAD(QC1,QC2); END;
        BEGIN READON(QI1,QI2); QI3:=EREAD(QI1,QI2); END;
        BEGIN READON(QA1,QA2); QA3:=EREAD(QA1,QA2); END;
        BEGIN READON(QN1,QN2); QN3:=EREAD(QN1,QN2); END;
        BEGIN READON(QIO11,QIO12); QIO13:=EREAD(QIO11,QIO12); END;
        BEGIN READON(QIO21,QIO22); QIO23:=EREAD(QIO21,QIO22); END;
        BEGIN READON(SC); WHILE SC¬=0 DO BEGIN TEM:=JOBTAB(0,0,0,0,0,0,0,
        0,0); EVBOSS(10,SC,TEM); READON(SC); END; END;
        BEGIN READON(P); GO TO HLT; END;
     END; GC TO START; HLT: END;

INTEGER PROCEDURE EREAD(INTEGER Q1; INTEGER VALUE RESULT Q2);
     CCMMENT**********************************************************
     * EREAD ASSIGNS A VALUE TO THE THIRD INPUT PARAMETER & READS IN   *
     * AN EMPIRICAL DIST IF THERE IS ONE.                              *
     **********************************************************;
     BEGIN INTEGER C; C:=0;
     IF (Q1=1)OR(Q1=3) THEN BEGIN READON(C); GO TC STP; END;
     IF Q1=4 THEN BEGIN READON(C); READ(QEMP(AC,1,1),QEMP(AC,1,2));
     FOR I:=2 UNTIL Q2 DO FOR J:=1 UNTIL 2 DO READON(QEMP(AC,I,J));
     Q2:=AC; AC:=AC+1; END;          STP:; C      END;
```

```
PROCEDURE WRITER;
     COMMENT**************************************************
     * WRITER SIMPLY WRITES OUT INPUT VALUES & APPROPRIATE DEFAULTS FOR*
     * ALL INPUT PARAMETERS.                                         *
     **************************************************************;
     BEGIN INTFIELDSIZE:=6;
   WRITE("***************JOB PARAMETERS***************"); WRITE(" ");
     WRITE("1=>UNIFORM...2=>EXPONENTIAL...3=>NORMAL...4=>EMPIRICAL...5=>
CONSTANT"); WRITE(" "); WRITE(" ");
     WRITE("FIXED INTERVAL JOB PARAMETERS"); WRITE(" ");
     WRITE("||PRIORITY(1)",QIP1,QIP2,QIP3,"||CORE(2)",QIM1,QIM2,QIM3,
     "||CPUTIME(3)",QIC1,QIC2,QIC3,"||IOTYPE(4)",QII1,QII2,QII3); WRITE
     (" "); WRITE("||ARRIVAL INTERVAL(5)",QIA1,QIA2,QIA3,
     "||NO OF REQUESTS(6)",QIN1,QIN2,QIN3); WRITE(" "); WRITE(" ");
     WRITE("FIXED FREQUENCY JOB PARAMETERS"); WRITE(" ");
     WRITE("||PRIORITY(7)",QFP1,QFP2,QFP3,"||CORE(8)",QFM1,QFM2,QFM3,
     "||CPUTIME(9)",QFC1,QFC2,QFC3,"||IOTYPE(10)",QFI1,QFI2,QFI3); WRITE
     (" "); WRITE("||ARRIVAL INTERVAL(11)",QFA1,QFA2,QFA3,
     "||NO OF REQUESTS(12)",QFN1,QFN2,QFN3); WRITE(" "); WRITE(" ");
     WRITE("BACKGROUND JOB PARAMETERS"); WRITE(" ");
     WRITE("||PRIORITY(14)",QP1,QP2,QP3,"||CORE(15)",QM1,QM2,QM3,
     "||CPUTIME(16)",QC1,QC2,QC3,"||IOTYPE(17)",QI1,QI2,QI3);WRITE(" ");
     WRITE("||ARRIVAL INTERVAL(18)",QA1,QA2,QA3,"||NO OF REQUESTS(19)",
     QN1,QN2,QN3); WRITE(" "); WRITE(" ");
     WRITE("||IOT1(20)",QIO11,QIO12,QIO13,"||IOT2(21)",QIO21,QIO22,
     QIO23); WRITE(" ");
     WRITE("************  C O N F I G U R A T I O N...CARD 13");
     WRITE(" ");
   WRITE("|RUNTIME LIMIT ",QGTLIM,"|NO OF CPUS",QCP,"|I/O SETUP O/H",
   QIOS,"|I/O RELEASE O/H",QICSTOP,"|FREQ FOR FFREQ",QFF); WRITE(" ");
     WRITE("|INTERRUPT MASK",QIMSK); WRITE(" ");
     WRITE("|TIME SLICES:"); FOR N:=1 UNTIL QCP DO WRITEON(QTSL(N));
    WRITE(" ");
     WRITE("||INITIAL I/O PARM(23)    ",P,"  OUTPUT LEVEL(13)",QICF);
     WRITE(" ");
     WRITE(" "); WRITE("DEFAULT DISTRIBUTIONS:   ");FOR I:=1 UNTIL 21 DO
     IF CCFLG(I) THEN WRITEON("   ",CHTR(I));
WRITE("*******************************************"); WRITE(" ");
     INTFIELDSIZE:=14;  END;


PROCEDURE SPEC (REFERENCE(JCBTAB) D);
     COMMENT* SPEC COMPUTES FFREQ SLIPPAGE EACH SECOND                *;
     BEGIN INTEGER JOD,SLP; JOD:=ENTIME(D);
     IF FFCNT<QFF THEN BEGIN SLP:=QFF-FFCNT; IF IOF>2 THEN WRITE
     ("@@@ FFREQ SLIP: ",SLP,"  SECOND # ",TRUNCATE(JOD/1000));
     FFSLIP:=FFSLIP+SLP; IF SLP>MAXFFSL THEN MAXFFSL:=SLP; END;
     JOD:=JOD+1000; ENTIME(D):=JOD; EVBOSS(8,JOD,D); FFCNT:=0; END;
```

```
COMMENT**********M A I N L I N E **********
* FIRST 3 LINES INITIALIZE 132-CHTR BLANK LINES AND AXIS FOR MAP. L &  *
* M INITIALIZE RANDOM. AC IS USED BY EREAD TO INPUT EMPIRICAL DISTS.   *
* LINES 5 & 6 INITIALIZE THE QUEUES WITH DUMMY BLOCKS.                 *
     ********** ********************************************************;
MAIN: ULIN:="_____"; STROTIM:=0; STRETIM:=650;
FOR J:=0 UNTIL 12 DO BEGIN AXIS(0|1):=" "; FOR JA:=1 UNTIL 9 DO
AXIS(10*J+JA|1):="_"; AXIS(10*J+10|1):="|"; END; BLANK(0|1):=" ";
M:=524287; L:=566387; FFCNT:=FFSLIP:=FISLIP:=MAXFFSL:=0; AC:=1;
ETOP:=EV(0,-1,NULL,NULL); ETOP:=EV(0,0,NULL,ETOP);
ITOP:=IOQ(NULL,NULL,NULL); CPUP:=RFP(NULL,NULL,NULL);

COMMENT.........INITIALIZE CHTR FOR WRITER (USED BY DEFAULT DIST SECT)*;
CHTR(1):="FINTPRI"; CHTR(2):="FINTCOR"; CHTR(3):="FINTCPU"; CHTR(4):=
"FINTIOT"; CHTR(5):="FINTARV"; CHTR(6):="FINTNOR"; CHTR(7):="FFRQPRI";
CHTR(8):="FFRQCOR"; CHTR(9):="FFRQCPU"; CHTR(10):="FFRQIOT"; CHTR(11):=
"FFRQARV"; CHTR(12):="FFRQNOR";CHTR(13):="MISC    "; CHTR(14):="BACKPRI"
; CHTR(15):="BACKCOR"; CHTR(16):="BACKCPU"; CHTR(17):="BACKIOT"; CHTR(18
):="BACKARV"; CHTR(19):="BACKNOR"; CHTR(20):="IOTYPE1"; CHTR(21):=
"IOTYPE2";

COMMENT....DEFAULT PARAMETERS;
QIP1:=5;QIP2:=5;QIC1:=5; QIC2:=100; QIM1:=5; QIM2:=20; QII1:=5; QII2:=2;
QIA1:=5;QIA2:=250;QIN1:=5;QIN2:=1;                    QFP1:=5;QFP2:=4;
QFM1:=5;QFM2:=20;QFC1:=5;QFC2:=100;QFI1:=5;QFI2:=3; QIOF:=2;
QFA1:=5;QFA2:=2;QFN1:=5;QFN2:=1;QGTLIM:=5000;QCP:=2;QIOS:=50;QIOSTOP
:=75;QFF:=5;QTSL(1):=100;QTSL(2):=100;QM1:=5;QM2:=60;QC1:=1;QC2:=1000;
QC3:=5000;QI1:=1;QI2:=1;QI3:=4;QP1:=1;QP2:=1;QP3:=5;QA1:=5;QA2:=500;
QN1:=5;QN2:=3;QIO11:=5;QIO12:=100;QIC21:=5;QIC22:=50;P:=1;
QIP3:=QIC3:=QIM3:=QII3:=QIA3:=QIN3:=QFP3:=QFM3:=QFC3:=QFI3:=QFA3:=QFN3:=
QM3:=QN3:=QIO23:=QIO13:=QA3:=QIMSK:=0;

FOR I:=1 UNTIL 22 DO CCFLG(I):=TRUE; READER; WRITER;
FOR KA:=1 STEP 5 UNTIL 126 DO   BLANK(KA|5):=ULIN; FOR K:=1 UNTIL QCP DO
 BEGIN CSTR(K):=BLANK; CFLG(K):=CPASSGN(K):=CREM(K):=0; END;
NCORE:=JOBTOT:=CPUTOT:=GTIME:=QTIME:=ATAT:=IOTIME:=0;
RLEN:=RSCT:=RFJS:=RJOBS:=ILEN:=ISCT:=IOJS:=IJOBS:=0; FSLT:=0;
LST:=STFLG:=IFLG(1):=IFLG(2):=IFLG(3):=IFLG(4):=0; IID:=800; MAXEPS:=0;
SSQTT:=SSQR:=SSQI:=SIGR:=SIGI:=ICT:=FCT:=BCT:=0;          IDF:=499;

COMMENT.. INITIALIZE EVENT LIST;
LB:=TRUNCATE(RANDOM(QIC1,QIC2,QIC3)); ACORE:=TRUNCATE(RANDOM(QIN1,QIN2,
QIN3)); TEM:=JOBTAB(800,TRUNCATE(RANDOM(QIP1,QIP2,QIP3)),TRUNCATE(RANDOM
(QIM1,QIM2,QIM3)),LB,TRUNCATE(RANDOM(QII1,QII2,QII3)),TRUNCATE(RANDOM
(QIA1,QIA2,QIA3)),ACORE,TRUNCATE(LB/ACORE),0);
EVBOSS(1,ENTIME(TEM),TEM);
LB:=TRUNCATE(RANDOM(QFC1,QFC2,QFC3)); ACORE:=TRUNCATE(RANDOM(QFN1,QFN2,
```

```
QFN3));TEM:=JOBTAB(500,TRUNCATE(RANDOM(QFP1,QFP2,QFP3)),TRUNCATE(RANDOM
(QFM1,QFM2,QFM3)),LB,TRUNCATE(RANDOM(QFI1,QFI2,QFI3)),TRUNCATE(RANDOM(
QFA1,QFA2,QFA3)),ACORE,TRUNCATE(LB/ACORE),0);
EVBOSS(1,ENTIME(TEM),TEM);
TEM:=JOBTAB(1,0,0,0,0,0,0,0,0); EVBOSS(1,3,TEM);
TEM:=JOBTAB(0,0,0,0,0,1000,0,0,0); EVBOSS(8,1000,TEM);
TEM:=JOBTAB(0,0,0,0,0,0,0,0,0); EVBOSS(9,1,TEM); EVBOSS(11,100,TEM);

LB:=ACORE:=0;
SCHEDULES; MAP(0,0,0,1); STAT;
COMMENT* NEXT STMT READS CARD TYPE 24.                                    *;
READ (L); READON(L); IF L=1 THEN GO TO MAIN;
END.
```

APPENDIX D. INPUT PARAMETERS

The purpose of this Appendix is to discuss in detail the use and preparation of the 24 types of input cards. It is assumed that Section A.11 has been studied. A number of general rules dealing with card preparation are listed first.

1. Each card type is identified by an integer as indicated in the LISTING BY CARD TYPE below. The first number to appear on every input card must be this card identification number.

2. With the exception of Card 23 and Card 24, the input cards may appear in any order in the input stream.

3. Card 23 and Card 24 must always appear in the input stream. All other cards are optional.

4. Card 24 must always be the last card in a particular set of input cards and it must always be preceeded by Card 23. Note that the input stream may contain more than one set of input cards; in this case Rule 4 applies to each set.

5. On most card types three integers are punched following the card type number. The cards for which this is true are identified in the listing by a list of parameter names, each beginning with "Q", immediately following the Card type. These numbers are parameters used to call the random number generator RANDOM, providing a variate for the job/configuration characteristic defined by the card type. The parameter names are the variable names to which the integers on a particular card will be assigned by the input procedure READER; they are lised to aid in associating this text with the program listing in Appendix C. As will be discussed shortly, the third parameter may

sometimes be omitted, depending on the distribution desired from RANDOM. These cards have the format <card type><Parm1><Parm2><Parm3>, where at least one blank must appear between each field. The formats for the three card types which contain data other than parameters for RANDOM are explained in the listing.

6. Whatever the format, the space on the card after the last number is not scanned by the program (but remember that every number for the program is followed by at least one blank) and may be used for comments that identify the card contents. This practice is particularly encouraged for Cards 23 and 24 since the order in which these cards appear is significant.

7. When reading the card descriptions below, remember that the Q-variables shown are parameters for RANDOM whereas the comments following the Q-variables apply to the variates returned from RANDOM. Thus making QIP1 larger does not guarantee higher priorities for FINT jobs; one must look at RANDOM (see Section A.12) to determine the effects of such a change.

LISTING BY CARD TYPE

-Card 1: FINT job priority (QIP1, QIP2, QIP3). Larger numbers imply higher priorities; FIFO within a priority.

-Card 2: FINT job memory requirement (QIM1, QIM2, QIM3). This parameter is to be used with a user-implemented core allocation routine; it does not affect the operation of the basic simulator.

-Card 3: FINT job CPU time (QIC1, QIC2, QIC3). Total CPU time requirement in milliseconds.

-Card 4: FINT job I/O type (QII1, QII2, QII3). Defines which of the four I/O "devices" will be used by each FINT job.

-Card 5: FINT job interarrival time (QIA1, QIA2, QIA3). The time between FINT job interrupts.

-Card 6: Number of FINT job I/O requests (QIN1, QIN2, QIN3). Determines the number of I/O requests each FINT job will make during its life in the simulator.

-Card 7: FFREQ job priority (QFP1, QFP3, QFP3). Larger numbers have higher priority, FIFO within a priority.

-Card 8: FFREQ memory requirements (QFM1, QFM2, QFM3). This parameter is to be used by a user-implemented core allocation routine; it does not affect the operation of the basic simulator.

-Card 9: FFREQ CPU time (QFC1, QFC2, QFC 3). Total CPU time requirement in milliseconds.

-Card 10: FFREQ I/O type (QFI1, QFI2, QFI3). Determines which I/O "device" each FFREQ job will use.

-Card 11: FFREQ interarrival time (QFA1, QFA2, QFA3). This is a dummy parameter in the basic simulator since FFREQ jobs are restarted by the system as rapidly as they exit.

-Card 12: Number of FFREQ I/O requests (QFN1, QFN2, QFN3). Defines the number of I/O requests each FFREQ job will make during its life in the simulator.

-Card 13: This card contains from 8 to 18 integers, depending on the configuration of the system. The numbers, in order, are:

13- The card type for this card.

QGTLIM (RUNTIME LIMIT) - The global time (milliseconds) at which the run is to terminate.

QCP - The number of central processors in the configuration. 1 to 10 inclusive.

QIOS- The overhead, in milliseconds, to be associated with returning to normal operation after an I/O interrupt.

QIOF- The output level (see Section A.13) desired. 0 to 3 inclusive.

QTSL(I)- This entry comprises a series of integers, one for each CPU, which indicate the time-slice size associated with the corresponding CPU. There are QCP of these parameters.

Note that the user is specifing on Card 13 the actual numbers that will be used in the simulation. In order to override any one of the defaults for these parameters the user must prepare a card containing all of the above Card 13 entries.

-Card 14: Background job priority (QP1, QP2, QP3). Larger numbers have higher priority, FIFO within a priority.

-Card 15: Background job memory requirement (QM1, QM2, QM3). This parameter is to be used by a user-implemented core allocation routine; it does not affect the operation of the basic simulator.

-Card 16: Background job CPU time (QC1, QC2, QC3). Total CPU time requirement in milliseconds.

-Card 17: Background job I/O type (QI1, QI2, QI3). Determines which I/O "device" each background job will use.

-Card 18: Background job interarrival time (QA1, QA2, QA3). Time between initiation of successive background jobs.

-Card 19: Number of background job I/O requests (QN1, QN2, QN3). Determines the number of I/O requests each background job will make during its life in the simulator.

-Card 20: I/O Type 1 processing time (QIO11, QIO12, QIO13). These parameters specify the distribution to determine the time to process I/O operations of type 1.

-Card 21: I/O Type 2 processing time (QIO21, QIO22, QIO23). Same as above for I/O type 2.

-Card 22: Statistics interval specification. This card contains, after the card number, a series of integers which will be interpreted as the global times at which the user desires to call procedure STAT. The series is arbitrarily long and must end with a zero. The times need not be in increasing order. Statistics always reflect cumulative values; it is not possible to reset the statistics gathering machinery during a run.

-Card 23: Output Control - This card and as many following cards as required contain a series of integers which tell the program at what global times the user desires certain output called flagged output to be printed (c.f. Section A.13).

-Card 24: Run Termination Card - The first non-blank character after the card number must be an integer. If this integer is 1 the program will assume that another complete set of input parameters follows, the entire simulator will be reset, the another run with new parameters will commence. If this integer is not 1 the program will halt.

APPENDIX E. SAMPLE OUTPUT

```
**************JOB PARAMETERS**************
1=>UNIFORM...2=>EXPONENTIAL...3=>NORMAL...4=>EMPIRICAL...5=>CONSTANT

FIXED INTERVAL JOB PARAMETERS
||PRIORITY(1)     5      5       0  ||CORE(2)      5      20       0  ||CPUTIME(3)      5     100       0  ||IOTYPE(4)      5
      2       0
||ARRIVAL INTERVAL(5)      5     250       0  ||NO OF REQUESTS(6)      5       1       0

FIXED FREQUENCY JOB PARAMETERS
||PRIORITY(7)     5       4       0  ||CORE(8)      5      20       0  ||CPUTIME(9)      5     100       0  ||IOTYPE(10)      5
      3       0
||ARRIVAL INTERVAL(11)     5       2       0  ||NO OF REQUESTS(12)      5       1       0

BACKGROUND JOB PARAMETERS
||PRIORITY(14)     1       1       5  ||CORE(15)      5      60       0  ||CPUTIME(16)      1    1000    5000  ||IOTYPE(17)      1
      1       4
||ARRIVAL INTERVAL(18)      5     500       0  ||NO OF REQUESTS(19)      5       3       0

||IOT1(20)      5     100       0  ||IOT2(21)      5      50       0
************  C O N F I G U R A T I O N...CARD 13
|RUNTIME LIMIT   5000  |NO OF CPUS      2  ||I/O SETUP O/H     50  ||I/O RELEASE O/H     75  |FREQ FOR FFREQ      4
||INTERRUPT MASK      0
|TIME SLICES:   100     100
||INITIAL I/O PARM(23)          1    OUTPUT LEVEL(13)      3

DEFAULT DISTRIBUTIONS:      FINTPRI   FINTCOR   FINTCPU   FINTIOT   FINTARV   FINTNOR   FFRQPRI   FFRQCOR   FFRQCPU   FFRQIOT
FFRQARV   FFRQNOR   BACKPRI   BACKCOR   BACKCPU   BACKIOT   BACKARV   BACKNOR   IOTYPE1   IOTYPE2
**************************************************

                    JOBSTART:          500
                  GETCPU:          500                2
OVRLAP:             2  CFLG            1          0  IFLG          0         0         0         0
                       MAP:          500                500                 0
                    JOBSTART:            1
RELCPU:           500          500                0                   102
OVRLAP:           102  CFLG            0          0  IFLG          0         0         0         0
                       IO:          500                152
OVRLAP:           152  CFLG            0          0  IFLG          0         0         1         0
                    RELIO:          500                202
OVRLAP:           202  CFLG            0          0  IFLG          0         0         0         0
                    JOBSTART:          800
                  GETCPU:          800                250
OVRLAP:           250  CFLG            0          1  IFLG          0         0         0         0
```

```
                    MAP:              800                 0                 800
FXIT.. JO8ID & CPUTOT ARE:            500                 200
                 JOBSTART:                500
                 GETCPU:           501                     277
OVRLAP:           277  CFLG                1                 1  IFLG                  0                 0                 0                 0
                    MAP:                   501                 501                 800
RELCPU:           800                  501                 800                 350
OVRLAP:           350  CFLG                1                 0  IFLG                  0                 0                 0                 0
RELCPU:           501                  501                 0                   377
OVRLAP:           377  CFLG                0                 0  IFLG                  0                 0                 0                 0
                    IO:                  800                 400
OVRLAP:           400  CFLG                0                 0  IFLG                  0                 1                 0                 0
                    IO:                  501                 427
OVRLAP:           427  CFLG                0                 0  IFLG                  0                 1                 1                 0
                 RELIO:                  800                 450
OVRLAP:           450  CFLG                0                 0  IFLG                  0                 0                 1                 0
                 RELIO:                  501                 477
OVRLAP:           477  CFLG                0                 0  IFLG                  0                 0                 0                 0
                 JOBSTART:                801
                 GETCPU:           801                     500
OVRLAP:           500  CFLG                1                 0  IFLG                  0                 0                 0                 0
                    MAP:                   801                 801                 0
                 JOBSTART:                  1
                 GETCPU:                    1                 503
OVRLAP:           503  CFLG                1                 1  IFLG                  0                 0                 0                 0
                    MAP:                   1                   801                 1
FXIT.. JO8ID & CPUTOT ARE:            800                 500
EXIT.. JO8ID & CPUTOT ARE:            501                 500
                 JOBSTART:                501
                 GETCPU:           502                     552
                          RFPQ:                  502                 4
RELCPU:           801                  801                 1                   600
OVRLAP:           600  CFLG                0                 1  IFLG                  0                 0                 0                 0
                 GETCPU:           502                     600
OVRLAP:           600  CFLG                1                 1  IFLG                  0                 0                 0                 0
                    MAP:                   502                 502                 1
RELCPU:             1                  502                 1                   603
OVRLAP:           603  CFLG                1                 0  IFLG                  0                 0                 0                 0
                    IO:                  801                 650
OVRLAP:           650  CFLG                1                 0  IFLG                  0                 1                 0                 0
                    MAP:                    0                  502                 0

500XXXXXXXXXXXXXXXX_____________________________501XXXXXXXXXXXXXXXX_____________________801XXXXXXXXXXXXXXXX502XXXXXXX
_____|_____|_____|_____|_____|800XXXXXXXXXXXXXXXX|_____|_____|_____|_____|_____|1XXXXXXXXXXXXXXXX|_____|_____|_____|
     0

**********************************
                    IO:                  1                   653
OVRLAP:           653  CFLG                1                 0  IFLG                  0                 1                 1                 0
                 RELIO:                  801                 700
OVRLAP:           700  CFLG                1                 0  IFLG                  0                 0                 1                 0
RELCPU:           502                  502                 0                   700
OVRLAP:           700  CFLG                0                 0  IFLG                  0                 0                 1                 0
                 RELIO:                     1                703
OVRLAP:           703  CFLG                0                 0  IFLG                  0                 0                 0                 0
                    IO:                  502                 750
```

APPENDIX F. RANDOM NUMBERS

The purpose of this Appendix is to describe the methods used by the program to produce random variates for the simulation and the statistical tests performed on these generators. See Ref. [10] for more details on the first three of these routines.

Uniform (a,b) routine: The Uniform generator uses a multiplicitive congruential method to produce random numbers in the interval (0,1). These numbers are then scaled into the desired interval (a,b). The uniform generator was subjected to the frequency test in the interval (0,1). Ten sub-intervals were used with sample sizes of 100, 300 and 500. A 95% confidence interval was placed around the mean for each sample size. The results of these tests are tabulated below. Chi squared(9) equals 16.919 at the 95% confidence level.

Sample Size	Sample Mean	Interval	Chi Sq.
100	.50231	.484-.516	6.5999
300	.50744	.489-.511	2.5999
500	.50248	.493-.507	2.4400

Exponential (MU) routine: The exponential generator uses the inverse transform method to produce exponential variates of arbitrary mean. This routine uses U(0,1) variates from the Uniform routine above. The exponential generator was tested with the frequency test in 20 intervals. Sample sizes of 500 and 1000 were run; the theoretical mean was 1.0. Chi squared (19) is 10.117 at the 95% confidence level.

Sample Size	Sample Mean	Sample Std Dev	Chi Sq
500	.9932	.9873	6.781
1000	.9973	.9951	7.376

Normal (EX,VAR) routine: The Central Limit Approach is used to produce these variates. Twelve U(0,1) variates from the Uniform routine are used to produce each Normal variate. The N(0,1) distribution was tested with the frequency test; sample sizes of 500 and 1000 with 20 sub-intervals were tried. Chi squared(19) is 10.117 at the 95% confidence level.

Sample Size	Sample Mean	Sample STD Dev.	Chi Sq.
500	.0542	.9762	20.598
1000	.0233	1.0039	24.487

Judging from the frequency histogram, the normal curve produced by this routine has irregularities at about ± 1 sigma from the mean. These results indicate that the normal generator is statistically deficient.

Empirical routine: To determine an empirical variate, the generator first draws from U(0,1). This number is then used to interpolate into the table as described in Section A.12 in Appendix A.

Constant routine: This code segment simply returns a real-valued constant which was provided by the user.

The tests done on the random variate generators are minimal. Real statistical significance of results should not be counted on from these generators. Serial correlation tests should by performed on all generators. Larger sample sizes should be tested. Although the method used to produce normal variates is admittedly approximate, the deviation from the theoretical results indicated in the above table casts doubt on the validity of the U(0,1) variates from the Uniform routine; consequently, these should be double-checked. It is recommended that

the normal generator be completely replaced. The user should not rule out the possibility that variate generators are available as library routines and could be used by the simulator.

APPENDIX G. STATISTICAL CALCULATIONS IN PROGRAM OUTPUT

This Appendix describes in detail the calculations performed by procedure STAT at each call. These calculations lead to the statistical output of the program. Each entry in this Appendix begins with the exact identifying phrase that appears in the output (see Appendix E).

-Global Time (Msec): A direct printout of GTIME at the time of the call to STAT.

-Total Jobs Completed: Each time EXIT is called, it increments a job counter (JOBTOT). This statistic is a direct printout of JOBTOT. EXIT also maintains job counters by job type; these counters are printed out with the labels "FINT", "FFREQ", and "BACKGROUND".

-Total CPU Time Used, Msec.: Each time GETCPU procures a processor for a job and creates a RELCPU block for that job, it adds the job's processing time to CPUTOT. If the job is interrupted, an appropriate amount is subtracted from CPUTOT. The statistic is a direct printout of CPUTOT.

-Total I/O Time, Msec.: Each time IO does an I/O operation for a job, the I/O time is added into IOTIME. The printed value is IOTIME.

-CPU Utilization,%: GTIME is multiplied by the number of CPUs to obtain total available processor time. This is divided into CPUTOT, rounded to the nearest one-tenth of one percent, and the result is printed.

-Total I/O-CPU Overlap, Msec.: I/O-CPU overlap is accumulated by the special procedure OVRLAP, which is called every time the state of any I/O device or any CPU changes. See Appendix C for details on the operation of OVRLAP. Its accumulator is printed out directly. Note

that overlap as defined here occurs at any point where at least one CPU is busy at the same time that at least one I/O device is busy. Four I/O devices are assumed to be on-line, even though only two time distributions are shared among them.

-Percentage Overlap: Overlap time is divided by CPUTOT; the result is rounded to the nearest hundreth of one percent and printed.

-Average Turnaround Time, Msec.: Each time EXIT is called, the routine computes the difference between GTIME and the time of entry of the calling job and accumualtes this quantity in ATAT. The time of entry is obtained from the calling job's table. ATAT is divided by JOBTOT and the result is printed out.

-Variance (Turnaround): Each time EXIT is called it accumulates in SSQTT the square of the difference between GTIME and the time of entry of the calling job. The variance in turnaround time is computed as (JOBTOT*SSQTT-ATAT**2)/(JOBTOT*(JOBTOT-1)) and printed.

-Throughput, Jobs/Min: JOBTOT is divided by the number of minutes in the run so far; this number is rounded to the nearest integer and printed.

-Total Number of FFREQ Slips: At the end of each second, procedure SPEC is called. This procedure computes the difference between the required number of completions (QFF) and the actual number of completions of FFREQ jobs. This difference is accumulated in FFSLIP and printed.

-Max Slip: The maximum slippage for FFREQ jobs over all 1-second intervals in the run is maintained in MAXFFSL and printed.

-Total Number of FINT Slips: A FINT slip occurs any time a FINT job gets removed from its processor and sent to the RFPQ. RFPQ accumulates these occurances in FISLIP and its value is printed by STAT.

-RFPQ-Avg No. Enqueued Jobs: Before every change in the length of any queue, the queueing routine accumulates in RFJS of IOJS the product of queue length and length of the interval since the last change. This sum is adjusted for the job-seconds accumulated from the last length change to the present time, divided by GTIME in seconds, and printed out. The value of RFJS or IOJS is then reset to its value at the last length change, in case STAT has been called in the middle of a run.

-*Sample Avg: Every 100 msec. procedure SAMPLE samples the RFPQ and IOWH queue lengths and accumulates these quantities and their respective squares in SIGR, SIGI, SSQR, and SSQI. Procedures RFPQ and IOWH record the total number of jobs which ever enqueue in either queue; variables RJOBS and IJOBS hold these numbers. SIGR is divided by RJOBS and the result printed to produce this cross-check on average queue length.

-*Sample Variance: The quantities explained above are combined according to the formula (RJOBS*SSQR-SIGR**2)/(RJOBS*(RJOBS-1)) and printed.

-*Avg Wait Time, Msec: Accumulated job-seconds are divided by the total number of jobs entering the queue; this value is printed.

The last four quantities are also computed and printed out for the IOWH. Starred quantities are printed only if the number of jobs queued at the time of the STAT call is greater than 1.

STAT also prints, at Output Level 3 (See Section A.13 of Appendix A), the contents of the RFPQ, IOWH and Event List (along with the number of the task or utility procedure to be called by SCHEDULES).

APPENDIX H. DEFAULT PARAMETERS

This Appendix lists the values that parameters will be given by the program if no input cards are supplied. Unless otherwise noted in the listing, all default distributions are constants.

Fixed Interval Jobs

Priority	5
CPU Time	100 msec
Core	20
I/O Type	2
Arrival Time	250 msec
No. of I/O Requests	1

Fixed Frequency Jobs

Priority	4
CPU Time	100 msec
Core	20
I/O Type	3
Arrival Time	2 msec
No. of I/O Requests	1

Background Jobs

Priority	U(1,5)
CPU Time	U(1000,5000)
Core	60
I/O Type	U(1,4)
Arrival Time	500 msec.
No. of I/O Requests	3

Configuration

Total Run Time	5000 msec (simulated)
No. of CPUs	2
I/O Setup O/H	50 msec
I/O Released O/H	75 msec
FFREQ Completion Frequency	5 jobs/sec
Time Slices	100 msec
I/O Type 1	100 msec
I/O Type 2	50 msec
Output Level	2
IOF	1
Interrupt Mask	0 (interrupts on)

BIBLIOGRAPHY

1. Ahrens, J.H., and Dieter, U., "Computer Methods for Sampling from the Exponential and Normal Distributions," Communications of the ACM, v. 15, p. 873-882, October, 1972.

2. University of Michigan Computer Center, Time Sharing Supervisor Programs, by M.T. Alexander, May, 1969.

3. Aron, J.D., "Real Time Systems in Perspective," IBM Systems Journal, v. 6, p. 49-67, 1967.

4. Computer Science Department, Stanford University, Algol W. (Revised), by H.R. Bauer, S. Becker, S.L. Graham, and E. Satterthwaite, September, 1969.

5. Burroughs Corporation, B5500 SIMULA: Users Manual, November, 1968.

6. Digest of Papers for the Sixth Annual IEEE Computer Society International Conference, September, 1972.

7. A/S Regnecentralen, RC4000 Software - Multiprogramming System, edited by P. Brinch Hansen, February, 1971.

8. Lampson, B.W., "A Scheduling Philosophy for Multiprocessing Systems," Communications of the ACM, v. 11, p. 347-360, May, 1968.

9. MacDougall, M.H., "Computer Systems Simulation: An Introduction," Computing Surveys, v. 2, p. 191-209, September, 1970.

10. Naylor, T.N., and others, Computer Simulation Techniques, p. 43-102, John Wiley and Sons, 1966.

11. Naval Ordnance Systems Command Contract No. N0017-70-C-4448, Technical Report on General Requirements for a Standard Ordnance Executive, by Systems Consultants, Inc., 1050 31st St. N.W., Washington, D.C. 20007, 12 August 1971.

DISTRIBUTION LIST

		No. of Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Professor G.L. Barksdale, Jr., Code 72 Bv Computer Science Group Monterey, California 93940	1
4.	LT William C. Regmund 612 Zoe Street Houston, Texas	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

GINATING ACTIVITY (Corporate author) aval Postgraduate School onterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

PORT TITLE

n Operating System Model for Real-Time Applications

SCRIPTIVE NOTES (Type of report and inclusive dates)

aster's Thesis; December 1972

THOR(S) (First name, middle initial, last name)

illiam Charles Regmund, Jr.

PORT DATE ecember 1972	7a. TOTAL NO. OF PAGES 82	7b. NO. OF REFS 11
ONTRACT OR GRANT NO. ROJECT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)	
	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

STRIBUTION STATEMENT

pproved for public release; distribution unlimited.

PPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

STRACT

The basic organization of an operating system is often obscured by the riad details of a particular implementation. In any context where the conceptual havior of such systems is of interest, it is therefore desirable to have at nd some device which is transparent to fundamental concepts but opaque to the tails. The purpose of this work is to develop such a device in the form of a pical model of certain operating system functions. In order to test the utility this model, it is translated into a computer simulation program. Since operating stems for real-time applications are of particular interest here, some features real-time systems that are built into the computer model are discussed. Finally, me applications of the program, and thus of the logical model, are suggested.

FORM 1 NOV 68 1473 (PAGE 1)

S/N 0101-807-6811

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Multiprocessing						
Operating System Model						
Simulation						
Real-Time						
Resource Scheduling